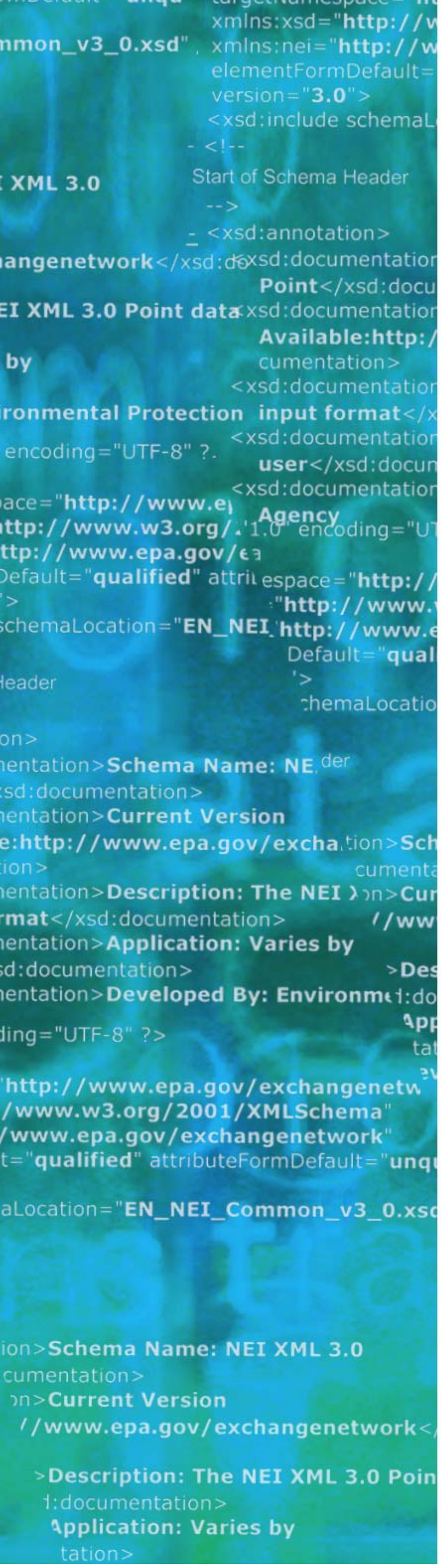


Schematron Validation and Guidance

Version:
1.0

Revision Date:
July, 18, 2007

Prepared for:
NTG

Prepared by:
Yunhao Zhang

Environmental Information exchange Network

SCHEMATRON VALIDATION AND GUIDANCE
VERSION 1.0

July, 20 2007

Revision History

Change Record			
Version Number	**Description of Change**	**Change Effective Date**	**Change Entered By**
1.0	Initial version	July 20,2007	Dr. Yunhao Zhang

Abstract

This document discusses the best practices for using Schematron rules for data validation within the Exchange Network. This Schematron guidance explains how Schematron can be utilized to validate data in an effective and fast way. Users should use this guide to develop Schematron rules for Nodes on the Exchange Network.

Table of Contents

1 Introduction

One of the key components in a business process is data validation. This is especially important in conducting data exchanges where data is exchanged between heterogeneous systems and databases. Traditionally, data validations are done using procedures written in programming languages such as C++, Java, or Visual Basic. Such procedures could become very complex and difficult to develop, debug and maintain when the type of document varies.

With the introduction of the Extensible Markup Language (XML), some of the business rules can be, at least partially, specified in XML schemas, and XML documents can then be validated against the schemas. Since XML is more restrictive than other markup languages, validation based on an XML schema can detect many data element errors such as type mismatches, missing elements and even referential integrity issues.

Since an XML Schema is not defined as a data validation mechanism, many of the business rules cannot be handled by simply using schema definitions. This is where Schematron comes into play as a data validation tool. Schematron is an ISO standard specifically defined for data validations.

Schematron has a number of advantages over traditional data validation methods, including the following:

- **Standard format**: Unlike validation in other programming languages, where a programmer determines how to use and enforce business rules using IF-THEN construct, Schematron rules follow a standard XML format. This allows sharing of Schematron rules cross different platforms.
- **Easier to develop and maintain**: As we will demonstrate shortly, Schematron rules are much closer to business rules and thus simpler to develop.
- **Removal of the traditional validation process**: Traditional data validation requires two basic elements, rules and the rule engine. The rule engine is basically the validation process, and the rules are the set of validation parameters. Due to a lack of a standard validation method, each business process often has its own data validation process. Schematron however, only requires users to develop business rules. The validation engine is the XSLT processor, which is available on almost all platforms.
- **More flexible and powerful**: Based on XPath and XSLT, Schematron allows the developer to randomly access any element in an XML instance document.
- **Highly extensible**: Using the extension mechanism in XPath, developers can add additional functions to support condition checking and assertions.
- **Descriptive Message**: Schematron allows developers to embed a natural language of error descriptions.

This document discusses the best practices for using Schematron rules for data validation within the Exchange Network. It also provides general guidance on the structure of business rules and format of the error messages.

2 Business Rules and Requirements

2.1 Schema and Schematron

An XML schema and Schematron can both be used to validate XML instance documents. XML schemas focus more on data type validations and data structures, while Schematron can be employed to enforce business rules. There are many business logic rules that cannot be expressed in terms of an XML schema construct. For example, a project ending date must be later than the project starting date, a facility ID must exist in a lookup table, or if element B is nonempty then element A must exist. These kinds of business rules are not directly supported by an XML schema, but are easily enforceable using Schematron.

The relationship between XML schemas and Schematron, in terms of data quality assurance, is complementary. Because XML schemas ensure basic data type correctness, Schematron validation should always be preceded by an XML schema validation.

2.2 Business Rule Definitions

Data validation requirements and business rules should be documented clearly before developing an XML schema and Schematron rules. The following table (Table 1) shows a recommended structure for defining business rules:

Table 1: Recommended XML schema and Schematron business rule definitions

Rule ID	Data Element	XML Element	Rule statement	Test Conditions	Error Level	Error Description	Validation Type
An identifier for the rule	The name of the data element	The name of the XML element	Technical description of the rule.	A list of test conditions	Level of error conditions: Warning, Error or Critical	A description of the error and how to fix it.	Either schema or Schematron

Each rule should have a unique ID within the rule set. It will be used in the error description by the Schematron rules.

The Rule Statement and Test Condition should contain enough information for developers to build assertions against the XML element. The Validation Type specifies whether the rule is checked by an XML schema or Schematron.

The following table (Table 2) is an example business rule:

Table 2: Example XML schema and Schematron business rule definitions

Rule ID	**Data Element**	**XML Element**	**Rule statement**	**Test Conditions**	**Error Level**	**Error Description**	**Type**
10	Site Terminated Date	SiteTerminatedDate	The date must be in YYYYMMDD format and in the range between 1/1/1959 and current date. It must also be later than the SiteEstablishedDate.	**Test 1**: Format: YYYYMMDD **Test 2**: Range: Jan. 1, 1957 <= X <= Current date **Test 3**: Threshold: SiteEstablishedDate < SiteTerminatedDate	Error	Use the rule statement.	Sche

3 Schematron Rule Development

3.1 Schematron Rule Elements

A Schematron rule contains three major elements:

1. **A Context**: The XML element which the rule applies to.
2. **An Assert or Report construct**: Test conditions the XML element must meet or violate respectively. This is usually an XPath expression.
3. **An Error Description**: Detailed description of the error and how to fix it.

A sample Schematron rule that verifies that the function ObservationDate complies with a set of business requirements is shown in the following table (Table 3) and definition example below:

Table 3: Sample Schematron rule with ObservationDate

Rule ID	**Data Element**	**XML Element**	**Rule statement**	**Test Conditions**	**Error Level**	**Error Description**	**Validation Type**
10	Observation Date	ObservationDate	The date must be in YYYYMMDD format and in the range between 1/1/1959 and current date.	**Test 1**: Format: YYYYMMDD **Test 2**: Range: Jan. 1, 1957 <= X <= Current date	Error	Use the rule statement.	Schematron

```
<rule context="aqs:ObservationDate">
      <assert test="neien:CheckDate(string(.),'', '19570101', 'Today',
0)"> [AQS23][Error]: <name zvon:fullPath='yes'/> ObservationDate
(<value-of select="."/>) must be in proper YYYYMMDD format and in the
range Jan 1, 1957 through today.
</assert>
</rule>
```

Note that a custom function, CheckDate, is used here to check the date format and date range in the sample. The rule uses a namespace prefix, AQS, in referencing the XML element ObservationDate. The explanation of how to define XML namespaces in the rule file is discussed next.

3.2 Using Namespaces

XML elements in Schematron rules should be fully defined with XML namespace prefixes. An XML instance document may contain or reference multiple namespaces. These namespaces must be defined in the Schematron rule file as shown by the following example:

```
      <ns prefix="MyPrefix"
uri="http://www.exchangenetwork.net/schema/MySchema/2" />
```

This example defines a prefix, MyPrefix, which is associated with the schema name space URI: http://www.exchangenetwork.net/schema/MySchema/2. Note that the namespace prefix is arbitrary, but the namespace URI must match exactly to what is defined in the schema file.

For the Exchange Network, we have defined a set of custom functions which support table queries, date verification, and regular expression checking. These custom functions are defined in a local namespace “`urn:neien-scripts`” (shown below).
If these custom functions are needed, you must add the following namespace definition in the Schematron rule file:

```
      <ns prefix="neien" uri="urn:neien-scripts"/>
```

To increase performance and efficiency, the QA server uses a compiled version of the same scripts which has a different namespace. To use the compiled version, the following namespace should be defined:

```
     <ns prefix="neien2" uri="urn:xmldata"/>
```

3.3 Error Message Format

The Schematron specification does not define an error message format. In the Exchange Network, however, we strongly recommend the following format be used for all error messages:

[ErrorType] [RuleId]: [Element Full Path] Error Description – Instruction

- **RuleId**: This should be the same business rule Id in the business rule table. A flow identifier should be attached if the ID is just an integer. For instance, the RuleId for rule 22 in NEI flow should be NEI22.
- **ErrorType**: This is the type of error, defined as Error, Warning, Critical, or some custom error level.
- **Element Full Path**: This is the complete path with the element name, leading to the offending element in the instance document. The complete path can be obtained using the `<name zvon:fullPath='yes'/>` construct (see example below).
- **Error Description**: This is the description of the error. This part must contain the offending value.

The following is an example error description in a Schematron rule:

```
[Error][AQS23]: <name zvon:fullPath='yes'/> ObservationDate (<value-of
select="."/>) must be in proper YYYYMMDD format and in the range Jan 1,
1957 through today.
```

The following example marks it as a Warning instead:

```
[Warning][AQS23]: <name zvon:fullPath='yes'/> ObservationDate (<value-
of select="."/>) must be in proper YYYYMMDD format and in the range Jan
1, 1957 through today.
```

The Warning error notice usually indicates that the XML element is optional, or the offending value is still acceptable by the receiver.

3.4 Schematron Software Developer Kit

To facilitate development of Schematron rules, we have created a Schematron SDK which contains the following components:

1. The basic Schematron v1.5 meta-files.
2. A compiled Microsoft XSL transformer.
3. A set of custom extensions to Schematron, including support of table lookup and date validation, etc.
4. Some sample Schematron files.

With this SDK, a developer can write Schematron rules on a desktop computer.

3.4.1 Rule Validation

Once the Schematron rules are created, you can test them using the following command in a command prompt:

```
msxsl.exe  YourSchematronFile zvonSchematron.xsl -o YourStyleSheet.xsl
```

Where zvonSchematron.xsl is the Meta style sheet for Schematron, which is included in the SDK. The command transforms the Schematron rule file to an XSLT style sheet.

If there is no error, you can then transform an XML document to an error file using the command below:

```
msxsl.exe  XMLInstanceDocument.xml YourStyleSheet.xsl -o ErrorFile.txt
```

This is often called the second phase transformation. The XML instance document is valid if the ErrorFile.txt is empty. Otherwise, the ErrorFile.txt should contain error and warning messages.

Note that any style sheet errors or system errors will be printed directly to the screen.

3.4.2 Debugging Schematron Rules

The best way to check if Schematron rules are functioning properly is to create an XML instance document that triggers each and every rule, and a clean XML document that passes all the rules.

Typically, if a rule could not be triggered, it could be one of the following issues:

- The context is incorrect.
- The namespace of the target XML element is missing or incorrect.
- The condition in the assertion always evaluates to true.
- The value in the XML document is correct.

One strategy in dealing with a faulty Schematron rule is to isolate it from all others. This can be done by creating a simple Schematron file that contain only the problematic rule, and then using it to check against an instance document.

4 Schematron Extensions

In order to perform operations that are not available in XPath, we created a set of extension functions. These functions support database queries, regular expressions, and string format validations. These are included in the custom.xsl file.

This section defines the custom functions and discusses their uses. Note that the custom function list may be subject to changes, and additional functions may be introduced in the future.

4.1 Regular Expression Support

4.1.1 Function

```
Public Function CheckExpression(pattern, value)
```

4.1.2 Parameters

- **Pattern**: The regular expression pattern a value must meet.
- **Value**: The value in the XML document to check against.

4.1.3 Description

This function checks a value in an XML document against a regular expression. It returns a true response if the value satisfies the expression, and returns a false response for values that don't satisfy the requirements.

This function uses a Microsoft RegExp object in Visual Basic. For information about regular expression syntax and usage, please refer to the Regular Expression Syntax and the Introduction of Regular Expressions.

4.2 Database Lookup

4.2.1 Function

```
Public Function CheckExist(dsn, sql, name, value)
```

4.2.2 Description

This function checks whether a value exists in a result set selected by an SQL statement. The function returns true if the specified value exists otherwise it will return a false.

Note that this function caches the result set if the name parameter is not empty.

4.2.3 Parameters

- **DSN**: The ODBC data source name. You can configure the DSN using the ODBC manager in the Control Panel.
- **SQL**: A SQL Query statement that retrieves a list of values from a database table. The select statement typically should contain a single column name.
- **NAME**: An identifier for the retrieved results. The SQL statement will only be executed once when the function is called the first time.
- **Value**: The value in an XML document to be verified.

4.2.4 Example

```
<assert test="neien2:CheckExist('NEILookup', 'select FIPS_ST_CTY from
StateCountyFIPSCode','FIPSCode', ‘NEI:CountyStateFIPSCode’)">
   [Error][NEI31]: <name zvon:fullPath="yes"/>: CountyStateFIPSCode has
an invalid value (<value-of select="."/>) according to the lookup
table.
</assert>
```

4.3 Current Date

4.3.1 Function

```
Public Function GetCurrentDate(format)
```

4.3.2 Parameter

Format: The format for the date. The default is
`%Y-%m-%d`

4.3.3 Description

The function retrieves the current date and returns it as a string.

4.4 Dynamic Date Validation

4.4.1 Function

```
Public Function CheckDate(dateValue, format, minDate, maxDate,
offsetValue)
```

4.4.2 Description

This function checks a date element in the XML document. It verifies that the date is in a specific format, and the date is within a specified range.

4.4.3 Parameters

- **DateValue**: The date value to be checked.

- **Format**: The valid format for the date. YYYY, MM, DD are used to represent year, month and day respectively.
- **MinDate**: The minimum date value. -1 means not specified and Today means the current date.
- **MaxDate**: The maximum date value. -1 means not specified and Today means the current date.
- **offsetValue**: The offset value of the Today's date. It can be negative or positive. The value is added to the maxDate or MinDate, if 'Today' is used.

4.4.4 Example

The following assertion makes sure the SiteTerminateDate is a valid date in YYYYMMDD format, and it is between 1/1/1957 and 1 year beyond today.

```
<assert test="neien:CheckDate(./aqs:SiteTerminateDate,'YYYYMMDD',
'19570101', 'Today', 10000)">
[Error][AQS21]: <name zvon:fullPath='yes'/> SiteTerminatedDate (<value-
of select="."/>) must be in proper YYYYMMDD format and in the range
Jan. 1, 1957 through 1 year beyond today and later than
SiteEstablishedDate (<value-of select="../aqs:SiteEstablishedDate"/>).
</assert>
```

Note that the offset is added to an integer expression of today's date in YYYYMMDD format. If today is 2/28/2007, then the integer expression is 20070228, it becomes 20080228 if 10000 is added.

4.5 Data Quality Assurance Services

The QA service is a set of web services for both XML schemas and Schematron validations. The services definitions are available at:

https://tools.epacdxnode.net/xml/validator.wsdl
https://tools.epacdxnode.net/xml/validatorex.wsdl

The services defined in the second WSDL file are more suitable for machine to machine data validations.

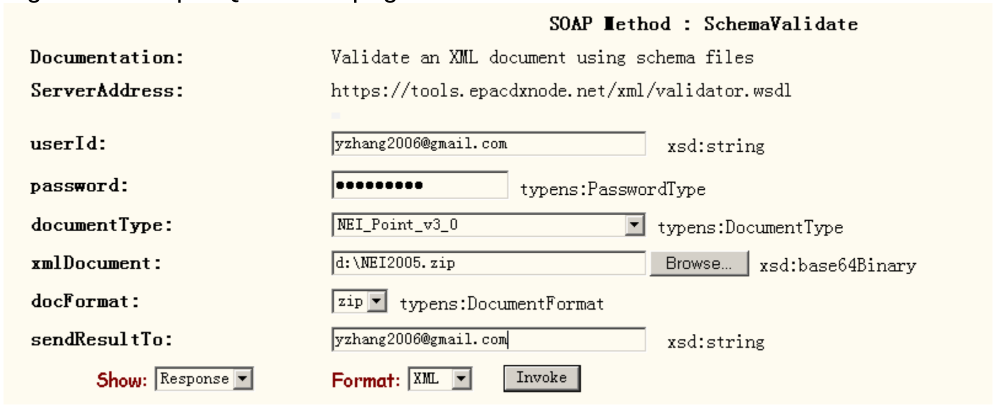
The QA server also provides a web interface for online real-time document validations which allows users to check document validity using a web browser. The following sections discuss how to use the browser tool.

The QA server web interface is available at https://tools.epacdxnode.net. Users of the tool must have a valid Network Authentication and Authorization Services (NAAS) account to access the services.

4.5.1 Using QA Server for Schema Validation

First type in the address, https://tools.epacdxnode.net, in a web browser URL, and click on the Retrieve button. You should see some machine generated web forms. One of the forms is called SchemaValidate, and is shown below (figure 1):

Figure 1: Sample QA server page for schema validation



SOAP Method : SchemaValidate

Documentation:	Validate an XML document using schema files	
ServerAddress:	https://tools.epacdxnode.net/xml/validator.wsdl	
userId:	yzhang2006@gmail.com	xsd:string
password:	••••••••••	typens:PasswordType
documentType:	NEI_Point_v3_0	typens:DocumentType
xmlDocument:	d:\NEI2005.zip	xsd:base64Binary
docFormat:	zip	typens:DocumentFormat
sendResultTo:	yzhang2006@gmail.com	xsd:string
Show: Response	Format: XML	

The QA server form is for Schema validation and it requires the following parameters:

- **userID/password**: This should be your NAAS account ID and password.
- **DocumentType**: The type of document to be validated. This document should be associated with an XML schema set. You can select the document type from the dropdown list.
- **xmlDocument**: The document to be validated. Click on the Browse button to select a document. This can be an XML document, or a compressed XML document in ZIP format. It is recommended that an XML document be compressed to save time and network bandwidth.
- **docFormat**: The format of your document. This can be in either an XML or ZIP format.
- **sendResultTo**: A valid email address where validation result can be sent. This parameter is required for large payloads.

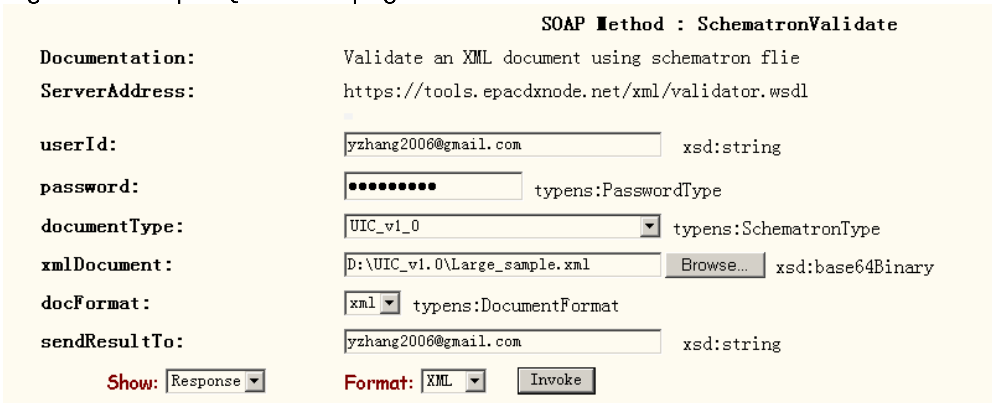
Click the Invoke button when all the parameters are entered. The server will return the validation result if the file size is relatively small. It returns a transaction ID if the document is too large to be handled in a short period of time. It will send the final result to the email address provided in such a case.

It is recommended that a document is verified using the Schema Validator first before using the Schematron Validator, which is discussed in the next sections.

4.5.2 Using QA Server for Schematron Validation

The same web page also contains a form for Schematron validation called SchematronValidate, and is shown below (figure 2):

Figure 2: Sample QA server page for Schematron validation



SOAP Method : SchematronValidate

Documentation:	Validate an XML document using schematron flie	
ServerAddress:	https://tools.epacdxnode.net/xml/validator.wsdl	
userId:	yzhang2006@gmail.com	xsd:string
password:	●●●●●●●●●	typens:PasswordType
documentType:	UIC_v1_0	typens:SchematronType
xmlDocument:	D:\UIC_v1.0\Large_sample.xml	xsd:base64Binary
docFormat:	xml	typens:DocumentFormat
sendResultTo:	yzhang2006@gmail.com	xsd:string

Show: Response Format: XML

The Schematron validation form has nearly the same parameters as the XML schema validation form:

- **userID/password**: This should be your NAAS account ID and password.
- **DocumentType**: The type of document to be validated. This is associated with a set of Schematron rules. You can select it from the dropdown list. The document type is not supported if it is not listed in the dropdown menu.
- **xmlDocument**: The document to be validated. Click on the Browse button to select a document. It can be an XML document or a compressed XML document in ZIP format. It is recommended that an XML document be compressed to save time and network bandwidth.
- **docFormat**: The format of your document. It can be in either XML or ZIP format.
- **sendResultTo**: A valid email address where validation result can be sent. This parameter is required for large payloads.

4.5.3 Deploying New Schematron Rules

[This section is for QA server administrators and operators]

The QA service is a generic web service that can support any Schematron validations without programming. However, a new set of Schematron rules must be configured before it can be made available to users.

The following is a simple process for deploying new Schematron rules:

1. Drop the new Schematron file to the SCHEMATRON subdirectory under the HDOC directory.
2. Add one entry to the server configuration file to publish it as a new documentType:
   documentTypeRules=/Schematron/***YourSchematronFile***

   Where documentType is the type of document to which the rules apply. It is typically the dataflow name.
3. Edit the validator.wsdl and validator.wsdl to add a new SchematronType. This will be displayed in the dropdown menu by the server.

4.5.4 Adding Database Lookup Functionality

As discussed in the previous sections, the Exchange Network Schematron Extension supports validating data from a lookup table using the CheckExist function. For this to work, ODBC connections must be created:

1. Go to the Control Panel and then the Administrative Tools.
2. Double click on the Data Sources icon. Under the System DSN click the Add button to add a new ODBC data source.
3. Choose a database driver as appropriate for you database.
4. Give a Data Source Name (DSN) for your database tables. For local lookup databases, you should also select the file that contains the lookup tables.

The DSN must match to what are being used in the Schematron rules.

5 References

- ***Schematron Specification and References***, http://www.schematron.com,
- ***Validating XML With Schematron***, Chimezie,Ogbuji, xml.com, Nov. 22, 2000.
- ***XML Path Language Version 1.0***, James Clark and Steve DeRose, W3C Recommendation, Nov. 16, 1999
- ***XSL Transformations Version 1.0.*** James Clark, W3C Recommendation, Nov. 16, 1999.