Data Validation in the NEMSIS XML Standard

Using the XML Schema and Schematron

Joshua Legler
November 2013
Welcome. This is a guide to data validation in the NEMSIS 3 XML standard. The purpose of this guide is to explain how the NEMSIS XML Schema and Schematron are used to validate EMS data.

The views expressed are those of the author (Joshua Legler) and do not represent the views of the NEMSIS Technical Assistance Center or any other entity.

It’s about a 20-minute read.
Here’s what we’ll cover:

• A refresher on XML basics
• Validation using the NEMSIS XML Schema
• Validation using Schematron
• Schematron rule design considerations
• Schematron performance
The NEMSIS standard is XML-based.

XML (eXtensible Markup Language) is a language that defines rules for encoding data using “markup.”

NEMSIS XML data look like this...
<?xml version="1.0" encoding="UTF-8"?>
<EMSDataSet xmlns="http://www.nemsis.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Header>
        <DemographicGroup>
            <dAgency.01>9999</dAgency.01>
            <dAgency.02>John's Ambulance Service</dAgency.02>
            <dAgency.04>10</dAgency.04>
        </DemographicGroup>
        <PatientCareReport>
            ...
        </PatientCareReport>
    </Header>
</EMSDataSet>
Software takes information entered by EMS personnel and transforms it into XML, like this...
PCR

Patient’s First Name:
John

Patient’s Middle Name:
V.

Patient’s Last Name:
Doe

...<ePatient>
  <ePatient.PatientNameGroup>
  <ePatient.02>Doe</ePatient.02>
  <ePatient.03>John</ePatient.03>
  <ePatient.04>V.</ePatient.04>
  </ePatient.PatientNameGroup>
  ...
  </ePatient>
  ...

XML
XML data can be validated.

NEMSIS 3 data are validated using:
- XML Schema
- Schematron
Here’s how XML Schema validation works...
First, software checks an XML document to make sure it is “well-formed.”

“Well-formed” means that it adheres to the basic rules of XML (elements are in markup <tag>s, every opening <tag> has a closing </tag>, etc.).
This XML document is *not* well-formed, because `<paragraph>` needs to have a matching `</paragraph>`.  

```xml
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <title>XML Sample</title>
  <paragraph>This is a sample XML document.</paragraph>
</document>
```
Second, if it is well-formed, the XML document is checked to make sure it is “valid.”

“Valid” means that it conforms to a Schema.

We often refer to an XML Schema as an “XSD.”
An XML Schema defines rules for what data an XML document can contain and how the document is to be structured.

An XML Schema is written in XML.
This XML document claims to conform to a Schema, and it lists the URL where the Schema can be found.

An XML validator can get the Schema from that URL and determine whether the XML document is valid...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<EMSDataSet xmlns="http://www.nemsis.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.nemsis.org http://nemsis.org/media/XSD_v3/_nemsis_v3.3.3/3.3.3.130926/XSDs/NEMSIS_XSDs_v3.3.3.130926/EMSDataSet_v3.xsd">
...
The NEMSIS Schema says this element:

- Can occur 0 or 1 time
- Is named “ePatient.03”
- Can be empty
- Can contain a string
- Can be 1-50 characters long

```xml
<xs:element
    id="ePatient.FirstName"
    minOccurs="0"
    name="ePatient.03"
    nillable="true">
  <xs:simpleType>
    <xs:restriction
        base="xs:string">
      <xs:minLength
          value="1"/>
      <xs:maxLength
          value="50"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
This piece of XML is not valid in the NEMSIS Schema, because the string in `<ePatient.03>` is longer than 50 characters.

...<ePatient.03>This Is A Patient First Name That Is 55 Characters Long</ePatient.03>...

...
Software developers use the information from the NEMSIS XML Schema to decide how to design their database. They’ll probably make the patient’s first name be a string of up to 50 characters in their database, since that’s what the NEMSIS Schema says.
Software developers also use the information from the NEMSIS XML Schema to build validation routines into their interface at the time of data entry...
A patient’s first name longer than 50 characters won’t fit in the database, and it won’t be valid in the XML output, so it makes sense to catch the problem as soon as possible...

**PCR**

Patient’s First Name:

This Is A Patient First Name That Is 55 Characters Long

Oops! Patient’s first name has to be less than 50 characters long.
Other validation checks have to wait until later:

Back in Service Date/Time cannot be empty—but we need to allow the EMT to finish the incident and come back to it at the end.
Even though the software tries to catch errors as early as it can, the Schema is the standard.

When it’s time to finish a patient care report (or later whenever it’s edited), the PCR is transformed into XML and checked to ensure it is both well-formed and valid.
Whenever an XML document is transferred from one system to another, the receiving system checks that the document is both well-formed and valid, to make sure nothing was broken in transit.

This also helps protect the receiving system from bad data that compromises its security.
Let’s recap what can be done with the NEMSIS XML Schema. The following things can be checked in an XML document by using the Schema...
• Well-formedness (adherence to the basic XML standard)
• Where an element is within a document
• How an element is grouped with others
• How many times an element can occur in a document

• Data types (string, integer, decimal, date, date/time, values from a defined list, etc.)
• Min and max values or lengths
• Pattern matching (using “regular expressions”)
The Schema does a lot, but it can’t do everything we need. For example, it can’t cross-check the data in one element against the data in another element:

• It can’t say that times (notified, en route, at scene, etc.) have to be in a certain order.
• It can’t say that the patient cannot be pregnant if the patient is male.
Enter Schematron...
Schematron is the tool for catching what the Schema can’t. The inventor of Schematron called it “a feather duster to reach the parts other schema languages cannot reach.”

Schematron can do everything the Schema can—but it wouldn’t make sense to put something in Schematron that’s already in the Schema.
Schematron is XML-based.

Once a PCR has been converted to XML to check that it is well-formed and valid according to the Schema, it’s also checked against a Schematron rules file.
Here’s how Schematron works...
A Schematron file mainly contains patterns, rules, and asserts:

```
<pattern id="(ID)">
  <title>(Subject: What this is about)</title>
  <rule context="(Context: Where to look in the document)">
    <assert test="(Condition: A true/false statement)">
      (Message to show if the test fails [test=false])
    </assert>
  </rule>
</pattern>
```
Here's a Schematron pattern for catching a pregnant male:

```xml
<pattern id="pregnant_male">
  <title>Males cannot be pregnant.</title>
  <rule context="PatientCareReport[eHistory/eHistory.18 > 3118001]" role="[FATAL]">
    <assert test="ePatient/ePatient.13 = '9906003'">
      Oops! Pregnant male.
    </assert>
  </rule>
</pattern>
```
As with the NEMSIS XML Schema, software developers can use the information from the Schematron file to build validation routines into their interface at the time of data entry:
An EMS unit should not be back in service before the call began, so it makes sense to catch the problem as soon as possible.

**PCR**

<table>
<thead>
<tr>
<th>Event</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notified</td>
<td>12:00</td>
</tr>
<tr>
<td>En route</td>
<td>12:04</td>
</tr>
<tr>
<td>On Scene</td>
<td>12:09</td>
</tr>
<tr>
<td>Left Scene</td>
<td>12:23</td>
</tr>
<tr>
<td>Destination</td>
<td>12:38</td>
</tr>
<tr>
<td>Back in Service</td>
<td>11:45</td>
</tr>
</tbody>
</table>

*Oops! Back in Service Date/Time is too early.*
Other validation checks have to wait until later:

Only females can be pregnant—but gender and pregnancy might be collected in different areas of the PCR, so we need to allow the EMT to get to both before we check.
Even though the software tries to catch errors as early as it can, the Schematron file is the standard.

When it’s time to complete a patient care report and the PCR is transformed into XML and checked to ensure it is both well-formed and valid, it is also checked against the Schematron file.
Whenever an XML document is transferred from one system to another, the receiving system checks that the document is not only well-formed and valid, but that it also follows the rules contained in the Schematron file.
Schematron is more flexible than the Schema...
Schematron can do “warnings” as well as “errors:”

• With the Schema, if an XML document is not well-formed or not valid, it should be rejected by the receiving system.

• But with Schematron, we may designate some rules as only “warnings” and not reject the document.
Schematron rule files can be updated more easily:

• Since the NEMSIS XML Schema defines the *structure* of an XML document, software developers have to reprogram their software whenever the Schema is changed.

• With Schematron, a state can add rules to their file and distribute it to software developers at any time.
Let’s recap what can be done with Schematron:
• Schematron can do everything the Schema can, but it’s generally used to do what the Schema cannot.
• It’s particularly good for cross-checking combinations of data within an XML document.
• It can do “warnings” as well as “errors.”
• Rules can be updated more easily over time.
Let’s look at some design considerations for Schematron rules...
Both the title and the message shown to the user need to be clear, concise, and actionable.

```xml
<pattern id="pregnant_male">
  <title>Males cannot be pregnant.</title>
  <rule context="PatientCareReport[eHistory/eHistory.18 > 3118001]" role="[FATAL]">
    <assert test="ePatient/ePatient.13 = '9906003'">
      Oops! Pregnant male.
    </assert>
  </rule>
</pattern>
```
The software needs to know which elements the user can fix. It can parse the Schematron file, or we can provide that information in the user message.

```xml
<pattern id="pregnant_male">
    <title>Males cannot be pregnant.</title>
    <rule context="PatientCareReport[eHistory/eHistory.18 > 3118001]" role="[FATAL]">
        <assert test="ePatient/ePatient.13 = '9906003'">
            Oops! Pregnant male. [ePatient.18] [eHistory.13]
        </assert>
    </rule>
</pattern>
```
Patterns, Rules, and Asserts should be designed to avoid the “one mistake triggers a dozen messages” problem.
Let’s recap Schematron design considerations:

• Text should be well-crafted.
• Software should know which elements the user can fix.
• One mistake shouldn’t trigger multiple error messages.
Lastly, let’s explore Schematron performance...
The first performance-related point:

XML is everywhere today, so lots of things are optimized to process it efficiently, including web browsers, mobile devices, and the tools used by software developers.
The NEMSIS national Schematron file for PCR data has 166 `<pattern>`s in it (November 2013).

Here’s how long it takes to process PCRs through that file...
Schematron Performance: PCRs

Saxon HE 9.5 (Java)
Intel Core i5 @ 2.60GHz

166 Schematron patterns
58ms overhead + 89ms per PCR

Seconds

PCRs
Let’s double the number of `<pattern>`s in the Schematron to 332.

Here’s how long it takes to process PCRs through that file...
Schematron Performance: PCRs

Saxon HE 9.5 (Java)
Intel Core i5 @ 2.60GHz

332 Schematron patterns
84ms overhead + 182ms per PCR
Based on these data, the marginal cost of adding one pattern to the Schematron file is:

\[
0.2\text{ms overhead} + 0.6\text{ms per PCR}
\]

When processing a single PCR, the marginal cost of a pattern is about one millisecond.
Those stats were produced on a laptop with an Intel Core i5 CPU at 2.60GHz running the Saxon XSLT processor in Java.

What about a smartphone? Roughly, multiply the time by 5–10x. With 332 patterns, a smartphone may need 1.3–2.6 seconds to process one PCR through Schematron.
Let’s recap Schematron performance:

• A 2013 model laptop requires about a quarter second to process one PCR through a Schematron file with 332 patterns.

• Each new pattern adds about one millisecond to the processing time.

• A smartphone may require 5–10 times as long to process a PCR through Schematron.
Let’s summarize this guide...
• NEMSIS is XML-based. All components of the standard are XML-based.
• Software developers build validation into their products based on the NEMSIS Schema and Schematron standards.

• When an agency’s demographic data is updated or a PCR is finished (or later changed), the software converts the data to NEMSIS XML and checks for well-formedness, validity, and compliance with the Schematron rules.
• XML validation and Schematron processing perform quickly on modern computers.
• Smartphones are slower but are capable of performing XML validation and Schematron processing. (However, software developers may choose to do it solely on the server.)
The purpose of this guide was to explain how the NEMSIS XML Schema and Schematron are used to validate EMS data.

Thanks for your attention!

Feel free to contact me at josh@joshuaalegler.com.