NEMSIS TAC Whitepaper

NEMSIS V3 Schematron Guide

Date

November 23, 2011 (Final)
January 17, 2014 (Rewritten – Candidate Release 1)
March 3, 2014 (Updated)
March 2, 2015 (Updated)

Authors

Joshua Legler – NEMSIS Consultant
Shaoyu Su – NEMSIS Software Developer
N. Clay Mann – NEMSIS P.I.

Contributors

Aaron Hart, Chris Morgan, Mike Darvill, and René Nelson – ZOLL
Adam Voss – TriTech Software Systems
David Saylor – Beyond Lucid Technologies
Jeff Robertson – EMSPIC
Jessica Lundberg – Cognitech
Juan Esparza – State of Florida
Kashif Khan and Troy Whipple – ImageTrend
Lindsey Narloch – State of North Dakota
Mark Potter – Medusa
Patrick Sennett – Good Samaritan Hospital
Paul Sharpe – Commonwealth of Virginia
Ryan Smith – Intermedix
Tom Walker – University of Alabama

Overview

Schematron is a rule-based language for XML document validation. Schematron is an international standard defined in ISO/IEC 19757-3(2006) (hereafter referred to as "normative standard"). Anyone who creates Schematron files or software that performs Schematron-based validation should obtain a copy of the normative standard at http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40833.

Much of the validation in NEMSIS is accomplished via the use of W3C XML Schemas (known as XSD). XML Schemas constrain the structure of NEMSIS XML documents and the contents of elements and attributes within those documents using grammar-based validation. However, XML Schemas are not capable of context-sensitive validation, such as constraining the contents of one element based on the contents of another element. The rule-based validation provided by Schematron fills the gap.

Purpose

This document sets forth requirements and guidelines for Schematron rule files and for software that performs validation using Schematron rule files. The following terms in this document have special meaning when displayed in uppercase:

- MUST: Specified feature is mandatory and will be verified through compliance processes.
- SHOULD: Specified feature is recommended but will not be verified through compliance processes.
- MAY: Specified feature is allowed but not mandatory.

Most of the content of this document is divided into the following two sections:

Structure of a Schematron File

Following is a general overview of the structure of a Schematron file. The normative standard should be consulted for details. *(? means the element must occur zero or one time; * means the element must occur zero or more times; + means the element must occur one or more times; no qualifier means the element must occur exactly one time.)*

`schema`	The document element for a Schematron file
`phase*`	A named group of patterns, to support phased validation
`active*`	A reference to a pattern that is active within a phase
`pattern+`	A container for related rules
`rule*`	A container for assertions tested within a particular context
`assert*`	An assertion (validation requirement) to be tested
text	A message to be generated if the assert test fails
`report*`	An assertion (validation requirement) to be tested
text	A message to be generated if the report test succeeds
`diagnostics?`	A container for diagnostic information
`diagnostic*`	A message giving more specific details concerning a failed assertion

Requirements and Guidelines for Schematron File Creators

NEMSIS Schematron File Content Requirements

All aspects of the normative standard apply to NEMSIS Schematron files, unless specifically constrained in this section. NEMSIS imposes additional requirements on the contents of Schematron files that are more restrictive than the normative standard. The additional NEMSIS requirements are listed below, by element.

`schema`

`@defaultPhase` MUST be undefined.

`@id` SHOULD be the name of the NEMSIS data set the schema is intended to validate (`DEMDataSet` or `EMSDataSet`).

`@queryBinding` MUST be `xslt2` (see XSLT 2 Query Language Binding, p. 4).

`@schemaVersion` SHOULD be the full NEMSIS release version and build number the schema is intended to validate (for example, `3.3.3.130926`), which MAY be followed by an underscore and a version number and build number specific to the Schematron file itself.

`title` SHOULD contain plain language and SHOULD indicate the name of the entity providing the schema (for example, the name of a state or territory).

`diagnostics` MUST exist and MUST at least contain the pre-defined `diagnostic` section supplied by the NEMSIS TAC. (See Referencing Diagnostics, p. 5.)

`pattern`

`title` MUST exist, SHOULD contain human-readable text, and SHOULD be written from the perspective of a business analyst.

`rule`

`@role` MUST NOT exist.

`assert and report`

`@diagnostics` MUST exist and MUST consist of a space-delimited list containing at least `nemsisDiagnostic`. (See Referencing Diagnostics, p. 5.)

`@role` MUST exist and MUST be `[FATAL]`, `[ERROR]`, or `[WARNING]`. (See Setting Severity Levels, p. 5.)

The content SHOULD be a natural language assertion written from the perspective of an EMS professional. (See Writing Natural Language Assertion Text, p. 7.)

Verifying Validity of Schematron Files

A NEMSIS Schematron file MUST be valid according to both the normative standard and the NEMSIS requirements. Specifically, it must be valid according to the following schemas:

- Normative standard RELAX NG schema for Schematron files
- Normative standard Schematron schema for Schematron files

- NEMSIS Schematron schema for Schematron files

All of the above schemas, as well as a NEMSIS Schematron file template designed to comply with NEMSIS requirements, are available from the NEMSIS TAC (see NEMSIS Schematron Development Kit, p. 10).

RELAX NG Schema validators are less prevalent than W3C XML Schema (XSD) validators; recommended tools include Jing (http://code.google.com/p/jing-trang/) and xmllint (http://xmlsoft.org/).

The following sequence of commands illustrates how to validate a NEMSIS Schematron file using Jing for RELAX NG validation and the Schematron reference implementation and Saxon (see Reference Implementation, p. 14) for Schematron validation (this is the process the NEMSIS TAC uses when validating files submitted by states):

First, process any Schematron `include` elements that are present in the Schematron file:

```
[path/to/saxon/]Transform
      -xsl:iso-schematron-xslt2\iso_dsdl_include.xsl
      -s:[SchematronFile.sch]
      -o:[SchematronFile1.sch]
```

Validate the Schematron file per the normative standard RELAX NG schema for Schematron files:

```
[path/to/jing]
      [path/to/]iso-schematron.rng
      [SchematronFile1.sch]
      > [RelaxNgValidationResults.txt]
```

Validate the Schematron file per the normative standard Schematron schema for Schematron files:

```
[path/to/saxon/]Transform
      -xsl:[path/to/]iso-schematron.xsl
      -s:[SchematronFile1.sch]
      -o:[IsoSchematronValidationResults.svrl]
```

Validate the Schematron file per the NEMSIS Schematron schema for Schematron files:

```
[path/to/saxon/]Transform
      -xsl:[path/to/]nemsis-schematron.xsl
      -s:[SchematronFile1.sch]
      -o:[NemsisSchematronValidationResults.svrl]
```

Schematron rule authors are welcome to submit Schematron code samples to the NEMSIS TAC early in their development cycle to obtain feedback regarding approaches they plan to use in writing Schematron files.

XSLT 2 Query Language Binding

In Schematron, the query language binding specifies the language used for rule context expressions, assertion tests, etc. Schematron implementations may support various query language bindings. However, NEMSIS Schematron files MUST use the Extensible Stylesheet Language Transformations (XSLT) version 2 query language binding, and all implementations MUST support XSLT 2.

XSLT 2 incorporates XML Path Language (XPath) version 2. Schematron rule authors should be proficient in XSLT 2 and XPath 2.

XSLT document() Function

The XSLT `document()` function allows a Schematron file to reference external resources. Schematron rule authors SHOULD avoid the use of the `document()` function, for the following reasons:

- If the `document()` function references an external resource, the Schematron file is no longer self-contained: it becomes dependent on the external resource. Systems that perform validation may not always be online and able to retrieve network-based resources, and they may not support the ability to locally manage external resource files along with a Schematron file.

- If the `document()` function references a relative URI or the Schematron file itself, the base URI may be ambiguous or undefined in systems that store and retrieve Schematron files from locations other than a file system (such as database or in-memory implementations).

Schematron rule authors should use the Schematron `include` element instead of the XSLT `document()` function. The `document()` function is evaluated each time validation is performed. On the other hand, the Schematron `include` element is evaluated at the time a Schematron file is compiled into XSLT when using the reference implementation of Schematron. (see Reference Implementation, p. 14).

Setting Severity Levels

Schematron rule authors MUST set a severity level for every `assert` or `report` by setting the `@role` attribute to `[FATAL]`, `[ERROR]`, or `[WARNING]`. Severity levels MUST NOT be set in any other location. The following examples demonstrate the three severity levels allowed in NEMSIS Schematron files:

```
<sch:assert role="[FATAL]"   diagnostics="…" id="…" test="…">…</assert>
<sch:assert role="[ERROR]"   diagnostics="…" id="…" test="…">…</assert>
<sch:assert role="[WARNING]" diagnostics="…" id="…" test="…">…</assert>
```

For information on how systems must behave when encountering the various severity levels, see Interpreting Severity Levels, p. 11.

Referencing Diagnostics

All Schematron files MUST contain a copy of the national `diagnostic` and MUST refer to it within `assert` and `report` elements. The national `diagnostic` ensures that detailed, structured information is available on every failed `assert` or successful `report`, which is used by software to provide a user with options for resolving validation problems. The `@id` of the national `diagnostic` is `nemsisDiagnostic`. The following example shows how the national `diagnostic` is referenced within an `assert`:

```
<sch:assert diagnostics="nemsisDiagnostic" role="…" id="…" test="…">…
</assert>
```

The national `diagnostic` contains three parts.

- `record`: A set of elements that uniquely identify the agency demographic report or patient care report where the validation problem was found.

- `elements`: A list of specific NEMSIS elements that were present in the record that the user may need to edit in order to resolve the validation problem.

- `elementsMissing`: A list of NEMSIS elements that were *not* present in the record that the user may need to edit in order to resolve the validation problem.

Every `assert` or `report` uses the `record` part simply by referring to the national `diagnostic`. Additionally, every `assert` or `report` SHOULD use the `elements` or `elementsMissing` part, or both parts, in order to facilitate the ability for the software to direct the user to the areas of the record that should be edited in order to resolve a validation problem.

Details on how to use each part are provided below.

`record`

Record information is generated for each failed `assert` or successful `report`. No further configuration is required on the part of the rule author.

`elements`

To use the `elements` part, the rule author must declare a Schematron variable within a `rule`. The variable MUST be named `nemsisElements` and MUST contain a set of XML elements defined using XPath, relative to the context of the `rule`. All elements in the set SHOULD be terminal elements: they SHOULD NOT contain any child elements.

The simplest example defines the `nemsisElements` variable as a set containing only the element that is the context of the current `rule`. If the context of the current `rule` is `eTimes.03`, the following will refer to an instance of `eTimes.03`:

```
<sch:let name="nemsisElements" value="."/>
```

The next example defines the `nemsisElements` variable as a set containing all of the children of the element that is the context of the current `rule`. If the context of the current `rule` is `eTimes`, the following will refer to all of the elements that are children of an instance of `eTimes` (`eTimes.01` through `eTimes.16`):

```
<sch:let name="nemsisElements" value="*"/>
```

The next example defines the `nemsisElements` variable as a set containing specific elements. If the context of the current `rule` is `eTimes` and the namespace prefix for the NEMSIS namespace is `nem`, the following will refer to `eTimes.03` and `eTimes.04` (the parentheses are important to indicate that the elements are part of a set):

```
<sch:let name="nemsisElements" value="(nem:eTimes.03, nem:eTimes.04)"/>
```

`elementsMissing`

An XPath reference cannot be generated for elements that *do not exist* in a particular record. The `elementsMissing` part is used to provide a list of names of missing elements, along with the XPath for the elements that would be the parents of the missing elements.

To use the `elementsMissing` part, the rule author must declare a Schematron variable within a `rule`. The variable must be named `nemsisElementsMissing` and must contain a space-delimited list of NEMSIS elements (without namespace prefix).

The rule author may also declare a Schematron variable named `nemsisElementsMissingContext`, which must contain a set of XML elements defined using XPath, relative to the context of the `rule`. Each element in the set should be an element that would be the parent of one or more of the elements listed in `nemsisElementsMissing`. If `nemsisElementsMissingContext` is not declared, then it is assumed to be the element selected by the context of the `rule`.

The first example defines the `elementsMissing` variable to be `eDispatch.03` (the single quotes inside of the double quotes are important to indicate that the `@value` is a string). If the context of the `rule` is `eDispatch`, the parent element is an instance of `eDispatch`:

```
<sch:let name="nemsisElementsMissing" value="'eDispatch.03'"/>
```

The next example is as above but defines the `elementsMissing` variable to be `eDispatch.03`, `eDispatch.04`, and `eDispatch.05`:

```
<sch:let name="nemsisElementsMissing"
  value="'eDispatch.03 eDispatch.04 eDispatch.05'"/>
```

The next example is as the first but explicitly sets the context to be an instance of `eDispatch`, if the context of the `rule` is `PatientCareReport` and the namespace prefix for the NEMSIS namespace is `nem`:

```
<sch:let name="nemsisElementsMissing" value="'eDispatch.03'"/>
<sch:let name="nemsisElementsMissingContext" value="nem:eDispatch"/>
```

The national `diagnostic` matches each element named in `nemsisElementsMissing` with its parent in `nemsisElementsMissingContext` by comparing element names. For example, `eDispatch.03` is matched with `eDispatch` because its name starts with `eDispatch`. If a missing element's name matches more than one of the elements in `nemsisElementsMissingContext`, the element that occurs first in the NEMSIS XML document is used. If a missing element's name matches none of the elements in `nemsisElementsMissingContext`, the element selected by the context of the `rule` is used.

Writing Natural Language Assertion Text

Natural language assertion text should be written from the perspective of an EMS professional and adhere to the following guidelines:

- It SHOULD consist of grammatically correct and complete sentences.
- It SHOULD use sentence case (not all UPPER or all lower case).
- It SHOULD be a positive statement of a constraint: it should be a statement of what is expected rather than a statement of what was found that was incorrect. The following examples illustrate the difference:

    - *Correct:* Unit En Route Date/Time should not be prior to Unit Notified by Dispatch Date/Time.
    - *Incorrect:* Unit En Route Date/Time is prior to Unit Notified by Dispatch Date/Time.

- It SHOULD use NEMSIS element names/titles ("Unit Notified by Dispatch Date/Time") rather than XML element names ("eTimes.03").
- When referring to expected enumerated list values, it SHOULD use NEMSIS value labels ("Yes") rather than XML values ("9923003").
- It SHOULD NOT have contents that are intended to be parsed by software, other than the Schematron elements allowed by the normative standard. Diagnostics should be used to provide such information.

Managing State and Local Rules

State and local rules MUST be provided in separate files from the national rules.

State and local rule files SHOULD avoid using `@id` values that are already used in the national Schematron files. Patterns, rules, asserts, and reports in the national Schematron files have `@id` values that start with `nemSch_`.

State rule files MUST contain an exact copy of the national `diagnostic` section and use it. (See Referencing Diagnostics, p. 5).

A state or local entity may escalate the severity level of an `assert` or `report` that is already defined in a national rule file. The recommended mechanism for doing so is to copy the national `assert` or `report` (and any elements on which it depends) into the state or local rule file and modify the `@role` attribute.

A state or local entity MUST NOT modify the national rule files. This implies that it is not possible for a state or local entity to de-escalate the severity level of an `assert` or `report` that is already defined in a national rule file.

States that maintain state rule files MUST submit the files to the NEMSIS TAC. The NEMSIS TAC will resolve any Schematron `include` elements and then validate the files. (See Verifying Validity of Schematron Files, p. 3.)

The NEMSIS TAC publishes the following valid and approved Schematron rule files for DEMDataSet and EMSDataSet:

- National rules
- State rules for each state that provides state rules

Versioning

New minor versions of the XML Schemas (XSDs) for NEMSIS are backward-compatible: any file that is valid in a previous version of NEMSIS will also be valid in the current version. In contrast, new versions of Schematron files for NEMSIS are not guaranteed to be backward-compatible: a file that is valid in a

previous version of NEMSIS may not be valid in the current version. The Schematron rule development process allows for new business constraints to be identified and introduced over time.

Schematron rule authors should consider how new rules may affect the validity of existing records. The following information is provided for consideration:

NEMSIS XML documents may contain `@xsi:schemaLocation`, which contains the URL of a specific release of the NEMSIS XML Schemas. Rule authors may choose to test the value of that attribute within rules and then apply assertions selectively based on the version to which the attribute refers. In doing so, rule authors would need to decide upon a default behavior in case the attribute is not present in the XML document.

New versions of the national rule files are published in accordance with the NEMSIS release cycle. When each new version of the NEMSIS standard is published for public comment, each state MUST test its state rule files to identify whether any incompatibilities exist. If incompatibilities exist, the state MUST submit updated state rule files to the NEMSIS TAC; otherwise, the state MUST notify the NEMSIS TAC that the existing state rule files are compatible with the new version of NEMSIS.

States may release updates to state rule files at other times outside of the NEMSIS release cycle. Each state SHOULD submit its release cycle to the NEMSIS TAC so that it can be published on the NEMSIS web site.

(See also Schematron Release Cycles, p. 13.)

Managing Optional Rules

States may define additional validation rules that are optional for use at the local level. Optional rules MUST NOT be included in state rule files. However, a state MAY include optional rules in a separate file and submit it to the NEMSIS TAC. The file MUST be a valid NEMSIS Schematron file. (See Verifying Validity of Schematron Files, p. 3.) The NEMSIS TAC does not review or endorse the contents of the file other than ensuring it is valid.

Maintaining Documentation

Good documentation is important for the successful implementation of validation rules. Schematron rule authors SHOULD document their rule files.

Documentation of the national rule files is published in a format similar to the NEMSIS Data Dictionary. The NEMSIS TAC publishes the same documentation interface for state rule files as well. Rule authors may strengthen the value of the documentation interface by the generous use of XML comments within their files. The comments should contain human-readable text regarding the elements within a file, including information such as why they are important, how they were developed, or pseudo-code representing how they function.

Designing for Performance

Schematron rule authors should consider the impact that each rule has on validation processing time. The following suggestions may help. While they are specific to the XSLT-based reference implementation of Schematron (see Reference Implementation, p. 14), they may apply to other implementations as well:

- Phased validation requires the overhead of validating a document in multiple passes. Phases are resolved in the process of compiling the Schematron file into XSLT, so a different XSLT file is generated to execute each phase of validation.

- Abstract patterns, abstract rules, and diagnostic references are resolved during the process of compiling the Schematron file into XSLT. Abstracts are good for maintainability and code re-use, but they do not improve performance.

- Processing is performed in document order. Each element in the document is selected one at a time. Then, each active pattern is searched for the first rule whose context matches the selected element. The first rule that matches is then fired, and all contents within the rule are processed. Performance is better if related rules are grouped together within a pattern, the context of each rule is more restrictive, and the rules that are more likely to be matched occur earlier within a pattern. Schematron files with fewer patterns and fewer *fired* rules will generally perform more quickly.

Consider using a predefined set of records for benchmarking. Each time an element is added or modified in a Schematron file, re-run the benchmark to calculate the performance impact of the change.

NEMSIS Schematron Development Kit

The NEMSIS TAC supplies a Schematron development kit. Its purpose is to help Schematron rule authors create well-designed Schematron files and validate them. It contains the following folders and resources:

- `rules`: Templates for building NEMSIS Schematron files

- `utilities/html`: an XSLT file and accompanying resources to generate documentation in HTML format from a Schematron file

- `utilities/schema`: RELAX NG and Schematron schemas for validating Schematron files

Requirements and Guidelines for Systems that Perform Validation

Schematron Validation within the Validation Workflow

All NEMSIS-compliant systems MUST perform XML Schema validation and Schematron validation:

- Systems that collect and send data MUST perform validation on each record at the time it is finalized.

- Systems that receive and process data MUST perform validation on each NEMSIS XML document that they receive.

Requirements for Schematron validation within the validation workflow are outlined in the Web Services Guide and the Compliance documentation. The following points are reiterated here:

All systems:

- A system MUST perform XML Schema validation before Schematron validation. If XML Schema validation fails, Schematron validation SHOULD NOT be performed.
- A system MUST be capable of performing Schematron validation on a NEMSIS XML document using multiple Schematron files (such as national, state, and local Schematron files). A system SHOULD process national rules first.

Systems that collect and send data:

- A system that collects and sends data MUST validate each record (agency demographic report or patient care report) when it is finalized (when data entry is completed by an EMS professional) and any time it is subsequently modified.

Systems that receive and process data:

- If XML Schema validation fails, the receiving system SHOULD reject the entire transaction.
- A receiving system MUST provide Schematron validation results using Schematron Validation Report Language (SVRL). (See Schematron Validation Report Language, p. 11.)

A Schematron file MAY define phases to support phased validation. A system MAY support selecting a phase for validation, as long as the system ensures that all patterns have been executed before a record is considered valid.

Schematron Validation Report Language

A system that receives and processes data MUST provide Schematron results using Schematron Validation Report Language (SVRL). SVRL is described in an informative appendix to the normative standard. The NEMSIS TAC maintains a modified version of the RELAX NG schema for SVRL in the NEMSIS Schematron Development Kit. In particular, the following deviations from the informative appendix are allowed in the NEMSIS schema for SVRL:

- `fired-rule` MAY occur zero times.
- `diagnostic-reference` MAY contain XML elements.

An SVRL document MUST be valid according to the NEMSIS RELAX NG schema for SVRL. (When using valid NEMSIS Schematron files, the XSLT-based reference implementation of Schematron generates valid NEMSIS SVRL output.)

Interpreting Severity Levels

Severity levels are defined using the `@role` attribute on `assert` and `report` elements.

A system MUST determine record validity based on severity levels in the following way:

- `[FATAL]`: If a record contains any `[FATAL]` problems, it is not valid.
- `[ERROR]`: If a record contains any `[ERROR]` problems, it is not valid.

- [WARNING]: If a record contains only [WARNING] problems, it is valid.

A web services transaction may include multiple records. A receiving system MUST behave as follows when receiving data, based on severity levels:

- [FATAL]: If a document contains any [FATAL] problems, it is not valid. The receiving system MUST reject the transaction.
- [ERROR]: If a document contains any [ERROR] problems, it is not valid. The receiving system MUST either reject the transaction or reject the records within the transaction that have [ERROR] problems while accepting the rest of the records.
- [WARNING]: If a document contains only [WARNING] problems, it is valid. The receiving system MUST accept the transaction.

Using Diagnostic References

Every valid NEMSIS Schematron file contains a diagnostic named nemsisDiagnostic. Systems that perform validation MUST implement nemsisDiagnostic. (If using the XSLT-based reference implementation of Schematron, this is accomplished by setting the parameter allow-foreign=true on the final stage of transforming a Schematron file into XSLT.)

When a document fails Schematron validation, every failed-assert or successful-report contains a diagnostic-reference with @diagnostic set to nemsisDiagnostic, with the following structure, in the http://www.nemsis.org namespace. *(? means the element must occur zero or one time; * means the element must occur zero or more times; + means the element must occur one or more times; no qualifier means the element must occur exactly one time.)*

nemsisDiagnostic

record	Elements that uniquely identify the record with the problem
dAgency.01	
dAgency.02	
dAgency.04	
eRecord.01?	Not present on demographics validation
elements	Elements that the user may edit to resolve the problem
element*	
@location	XPath expression
(any attributes that exist on the element)?	
(text)?	The value of the element
elementsMissing	Missing elements that the user may edit to resolve the problem

`element*`

`@parentLocation`	Xpath expression for the parent element
`@name`	XML name of element
(no text)	

The `nemsisDiagnostic` information enables a system to provide intelligent guidance to a user regarding which elements the user may need to edit in order to resolve a problem. Systems that collect and send data SHOULD process the diagnostic data and MAY highlight or provide links to the relevant elements of the agency demographic report or patient care report where the problem can be resolved.

Improving Performance

Schematron validation can generate verbose output in SVRL. The following suggestions may help. While they are specific to the XSLT-based reference implementation of Schematron, they may apply to other implementations as well:

- For production environments, the parameter `generate-fired-rule=false` may be set on the final stage of transforming a Schematron file into XSLT. SVRL output generated by that XSLT will not contain any `fired-rule` elements, which usually constitute the bulk of an SVRL document; it will still contain `failed-assert` and `successful-report` elements in cases where there are validation problems. The SVRL document will be considerably smaller, especially when there are no validation problems.

- Systems that receive and process data MAY omit Schematron processing results in their response to the client if all records in the transaction were accepted.

- Because of the implementation of severity levels within NEMSIS Schematron validation, a NEMSIS document may trigger `failed-assert` or `successful-report` statements in SVRL validation results and yet be a valid document (in other words, all of the `failed-assert` and `successful-report` statements may be at the `[WARNING]` level). The following XPath query of the SVRL output tests whether the document contains any validation problems at the `[FATAL]` or `[ERROR]` levels: `//@role=('[FATAL]','[ERROR]')`.

Schematron Release Cycles

New versions of the national rule files are published in accordance with the NEMSIS release cycle. Changes to the NEMSIS standard that are published in March MUST be implemented in all systems by the end of the year. (See “NEMSIS Version 3 Minor Release Schedule” on the Version 3 – Overview page of the NEMSIS Web site.)

Maintainers of state systems may choose to implement XML Schema changes quickly (to ensure that new data values are accepted in the state system) but defer Schematron changes until the end of the year (to ensure that the new rules have been implemented at the local level first).

If a state system implements new Schematron rules before a local system does, then a record that was found valid by a local system may be found invalid by the state system at the time of submission. The local system MUST communicate the error(s) to a user for resolution.

(See also Versioning, p. 8.)

Reference Implementation

A reference implementation of Schematron is available at http://code.google.com/p/schematron/. It implements Schematron processing using a series of Extensible Stylesheet Language Transformations (XSLT). NEMSIS Schematron files use XSLT2 as the required query binding and therefore require an XSLT processor that implements XSLT version 2.0. A schema-aware (SA) processor is not required.

The following sequence of commands illustrates how to transform a Schematron file into an XSLT file using the Saxon XSLT processor. (The commands are broken into multiple lines for clarity but should be typed on one line.)

```
[path/to/saxon/]Transform
      -xsl:iso-schematron-xslt2\iso_dsdl_include.xsl
      -s:[SchematronFile.sch]
      -o:[SchematronFile1.sch]

[path/to/saxon/]Transform
      -xsl:iso-schematron-xslt2\iso_abstract_expand.xsl
      -s:[SchematronFile1.sch]
      -o:[SchematronFile2.sch]

[path/to/saxon/]Transform
      -xsl:iso-schematron-xslt2\iso_svrl_for_xslt2.xsl
      -s:[SchematronFile2.sch]
      -o:[SchematronFile.xsl]
      allow-foreign=true
```

The first command processes any Schematron `include` elements that are present in the Schematron file. (Note that this first step has already been performed on all Schematron files published by the NEMSIS TAC.)

The second command expands all instances of abstract patterns. (See Designing for Performance, p. 9.)

The third command transforms the Schematron file into an XSLT file. The `allow-foreign=true` parameter ensures that non-Schematron elements in NEMSIS Schematron files are preserved (which is important for NEMSIS diagnostics to work). Other parameters are available as well and are documented in `iso_svrl_for_xslt2.xsl`. In particular, the `generate-fired-rule=false` parameter may be used to generate an XSLT file that will not produce `fired-rule` statements in its SVRL output. (See Improving Performance, p. 13.) If using phases, it is important to note that `phase` is provided as a parameter at this stage of processing. For example, if a Schematron file defines multiple phases, and an implementation intends to execute validation in phases (rather than the default behavior of executing all patterns at once), then a separate XSLT file will be created for performing each phase of validation.

Implementations based on the reference implementation should execute all of the above commands upon receipt of a new Schematron file. They can then use the resulting XSLT file to validate NEMSIS XML documents. The following command performs Schematron validation on a NEMSIS XML document and generates results in SVRL:

```
[path/to/saxon/]Transform
      -xsl:[SchematronFile.xsl]
      -s:[NemsisXmlFile.xml]
      -o:[ValidationResults.svrl]
```

In SVRL generated by the reference implementation, `active-pattern` elements contain a `@document` attribute, which reports the location (URI or file path) of the XML document being validated. The NEMSIS TAC modified version of the RELAX NG schema for SVRL allows but does not require the `@document` attribute. For security, it may be advisable for implementations based on the reference implementation to omit or suppress the `@document` attribute (it can be done by modifying `iso_svrl_for_xslt2.xsl` or post-processing its output).

The examples above demonstrate how to use Saxon to execute the reference implementation of Schematron via a command line. However, implementers should use tools and application programming interfaces (APIs) that are available within their development environment. For example, Saxon provides Java and .Net APIs.

Limitations

A combination of XML Schema validation and Schematron validation covers the majority of validation rules that may be identified in NEMSIS data. However, neither form of validation is well-suited for the following kinds of validation:

- *Authenticating the credentials of a web services client and determining whether the client is authorized to submit the data contained in a transaction.* These activities should be part of the security layer of a system that receives and processes data.

- *Validating data based on value lookups in large lists, such as GNIS, ICD-10, SNOMED, or RxNorm.* Schematron is capable of these activities, but it may not perform well on lower-performing hardware, and it may be difficult to keep the lists up-to-date in field installations. It is best suited for server processing.

- *Performing statistical or longitudinal analysis and validation.* For example, it is reasonable to assert that the gender mix within a large set of patient care reports should be close to 50/50; but since NEMSIS data may be transferred in batches as small as a single record, it is not practical to apply such an assertion to each transaction.

Conclusions

Schematron facilitates rule-based validation capabilities on NEMSIS data. The requirements in this guide ensure consistency and compatibility among NEMSIS Schematron rule files and software that performs validation on NEMSIS data.