# WIPO ST.96 XSD Validator Manual

Version 4.0

April 1, 2025

**Version History**

|  |  |  |
| --- | --- | --- |
| **Change date** | **ST96XSDValidator**  **version** | **Description of changes** |
| 2016-06-01 | 1.0 | Release based on WIPO Standard ST96 Annex I V2.1 |
| 2020-05-26 | 1.1 | Release based on WIPO Standard ST96 Annex I V4.0 |
| 2021-10-01 | 2.0 | Release to incorporate improvements and new spell check capability |
| 2023-03-29 | 2.1 | Release based on WIPO Standard ST96 Annex I V7.0 and minor improvements |
| 2023-09-22 | 2.2 | Release to fix bugs |
| 2024-09-05 | 3.0 | Release based on WIPO Standard ST96 Annex I V8.0 and improvements to integrity check, and conformance checks for GD-23, GD-24, GD-25, SD-01, SD-02, SD-04, SD-22, SD-34, SD-36, SD-40, SD-43, SD-47, SD-57, SD-62 and SD-63 |
| 2025-04-01 | 4.0 | Release based on WIPO Standard ST96 Annex I V9.0, which updates SD-03, and numerous improvements. See release notes. |

**TABLE OF CONTENTS**

WIPO ST.96 XSD Validator Manual 1

Introduction 3

1.1 Purpose 3

1.2 Benefits 3

1.3 Scope 3

Validating Conformance to ST.96 DRCs 3

About Schematron 4

Installing and Configuring ‘ST96XSDValidator’ 5

1.4 Requirements 5

1.5 Installation 5

1.6 IP Office Specific Configuration 5

1.7 Approved Exceptions Configuration 6

1.7.1 Approved Exceptions Update via Excel 7

1.8 Spell Check Configuration 7

Running ‘ST96XSDValidator’ 9

1.9 From the command line 9

1.9.1 Checking Schema(s) in a Directory 9

1.9.2 Checking Schema(s) in a Directory, by comparing against a previous version of the Schema(s) 11

1.10 With a Graphical User Interface: 12

1.11 Integrity checks 12

1.12 Component Name Report 12

1.13 Referenced Component Lookup 13

1.14 Unit Tests 14

1.15 Using oXygen XML Editor 16

1.15.1 Checking a Single File 17

References 18

Appendix A: Rules Enforced by the Schematron 19

Appendix B: Unit Testing Framework Details 36

# Introduction

## Purpose

This document describes ST96XSDValidator, a tool that uses Schematron to validate XML Schemas (XSDs) against *WIPO Standard ST.96, Annex I - IP XML Design Rules and Conventions (DRCs)*.

## Benefits

Automated enforcement of *WIPO Standard ST.96, Annex I - IP XML Design Rules and Conventions:*

* promotes interoperability and sound data exchange by enforcing a standard set of rules and guidelines,
* ensures consistency throughout the intellectual property communities and helps to accommodate growth and change by enforcing flexible XML design principles,
* decreases the level of effort required for XML developers to follow the standards by providing automated enforcement for most rules and a clear checklist for rules that must be manually checked.

## Scope

The scope of this document includes validation of conformance WIPO Standard ST.96 rules pertaining to the creation of XML Schemas. Rules pertaining to the creation of XML document instances are out of scope.

# Validating Conformance to ST.96 DRCs

*WIPO ST.96 XML Design Rules and Conventions* defines a comprehensive set of constraints to be followed when creating XSDs related to patent, trademark, design, geographical indication, and copyright Industrial Property (IP) types. XSDs should be validated to determine whether they conform to these rules. Sometimes this validation process can be automated, but in other cases it requires manual assessment.

An automated tool can enforce rules that are expressed mechanistically. For example, a rule requiring the length of component names to be less than 35 characters (GD-07) can be automated. Automation can enforce rules that state what is prohibited. For example, a rule stating that schemas must not use xsd:redefine (SD-07) can be automated.

An automated tool *cannot* enforce rules requiring human judgement. For example, a rule calling for names to be self-explanatory (GD-08) cannot be automated. Automatation also cannot enforce rules that state what is permitted rather than what is prohibited. For example, a rule stating that abstract types may be used (SD-47) involves nothing to check because it declares only that which is already allowed by W3C XML Schemas.

Appendix A has a table describing whether automation is possible and whether enabling assumptions are required each XSD rule in *WIPO ST.96 XML Design Rules and Conventions*.

This tool is intended to be run on design schemas, but may also be run on flattened schemas with the understanding that some rules may not apply to flattened schemas, or cannot be checked.

# About Schematron

Schematron is an ISO Standard (ISO/IEC 19757-3:2006) schema language that allows for the expression of constraints on XML documents. While most grammar-based schema languages (such as W3C XML Schema) can specify the general structure and valid values of an XML document, Schematron is much more flexible in terms of the kinds of constraints or "business rules" it can enforce. It uses XPath to express these constraints, and provides a custom vocabulary that helps to document the individual rules and the desired reporting procedures for violations of these rules.

The following code sample shows a Schematron pattern that enforces GD-07 (*the maximum length of a component name SHOULD be no more than 35 characters*):

<pattern>

<title>GD-07</title> ①

<rule context="② xsd:complexType[@name]

| xsd:simpleType[@name]

| xsd:element[@name]

| xsd:attribute[@name]">

<assert test="string-length(@name) &lt;= 35" ③

flag="AUTO" ④

role="WARNING" ⑤

diagnostics="componentName namespaceName uri" ⑦

see="#GD-07" ⑧>

⑥ The length of the

<value-of select="local-name(.)"/>

named

<value-of select="@name"/>

is

<value-of select="string-length(@name)"/>.

</assert>

</rule>

</pattern>

The major parts of this Schematron pattern work together to enforce the GD-07 rule:

① This pattern’s **title** specifies the rule identifier, GD-07, implemented by this pattern.

② The **context** of this rule specifies that GD-07 applies to named components (complexTypes, simpleTypes, elements, and attributes) in the input XSD.

③ The **assertion test** specifies that GD-07 requires names to be less than 35 characters long.

④ The **assertion flag** indicates that this rule can be determined AUTOmatically.

⑤ The **assertion role** indicates that the user should be WARNed if this assertion’s test proves not to be true. Violation of any rule using the SHOULD keyword results in a warning; violation of any rule using the MUST keyword results in an error.

⑥ The **assertion violation message** specifies the warning to display to when this assertion’s test fails.

⑦ The **diagnostics** references elements which provide diagnostic information such as the path of the input XSD, and is used to produce the Summary reports.

⑧ The **see** specifies the rule identifier, and is used for aggregating results in the Summary reports.

For a general introduction on ISO Schematron see Schematron Implementation and the Schematron & SVRL Cheat Sheet.

# Installing and Configuring ‘ST96XSDValidator’

## Requirements

While, it is recommended to have Java JDK 1.8 or later, the only external dependency required to run ST96XSDValidator is Java JRE 1.8 or later. Everything else is included in the installation zip file.

The XSD validator will not work with JDK 15 or later.

## Installation

1. Java JRE is sufficient for running the Validator, but the Validator will warn about the missing file "tools.jar" (this file is not needed to run the Ant tasks used by the Validator).
2. Download ST96XSDValidator\_*version*.zip.
3. Unzip ST96XSDValidator\_*version*.zip into an installation directory.
4. Update ST96XSDValidator/dev/settings.xml file with the Java JDK installation path from step #1. For example:

<JDKHome>”C:\Program Files\Java\jdk1.8.0\_11”</JDKHome>

or

<JDKHome>"C:\Program Files (x86)\Java\jre1.8.0\_291"</JDKHome>

The ST96XSDValidator is now ready to run. The remaining subsections in Section 0 describe optional configuration settings. You may skip ahead to Section 0 if you wish to run ST96XSDValidator in its default configuration.

## IP Office Specific Configuration

This section describes *optional* settings for specifying IP Office specific configuration. It is not required in order to use ST96XSDValidator.

Most ST.96 rules apply equally well to ST.96 XSDs and IPO-specific XSDs. The validator may be run without change on IPO-specific XSDs, and most of the rules will be checked just fine. For example, GD-07 checks that the maximum length of component names does not exceed 35 characters. This check will apply equally well to IPO-specific XSDs.

However, a rule such as SD-03, which says that Patent schemas must not refer to Trademark schemas (and vice-versa), can also be made to work for IPO-specific XSDs by providing IPO-specific namespace configuration information in the file, BaselineNamespaces.xml.

Simply add Namespace elements for any IPO-specific namespace information you wish to be considered during validation. This is entirely optional; validation does not require that any IPO-specific configuration be made.

Here are examples of IPO-specific namespace information configured in BaselineNamespaces.xml:

<Namespaces>

...

<Namespace subdir="USCommon" ns="urn:us:gov:doc:uspto:common" ns-prefix="uscom" std="USPTO" component="common"/>

<Namespace subdir="USPatent" ns="urn:us:gov:doc:uspto:patent" ns-prefix="uspat" std="USPTO" component="patent"/>

<Namespace subdir="USDesign" ns="urn:us:gov:doc:uspto:design" ns-prefix="usdgn" std="USPTO" component="design"/>

<Namespace subdir="USTrademark" ns="urn:us:gov:doc:uspto:trademark" ns-prefix="ustmk" std="USPTO" component="trademark"/>

</NamespacesNamespaces>

The purpose of BaselineNamespaces.xml is to centralize namespace settings in one location apart from the Schematron source code, thereby keeping the Schematron source IPO-neutral.

## Approved Exceptions Configuration

This section describes *optional* settings for approving exceptions to normal ST.96 rules checking. It is not required in order to use ST96XSDValidator.

Some rules violations may be decided to be acceptable. Rather than having acceptable violations forever appear on the summary report, it is possible to approve an exception to a ST.96 rule so that future runs of ST96XSDValidator will not report the violation.

For example, consider GD-14:

Acronyms and abbreviations MUST appear as listed in Appendix C for element and type names.

The heuristic used by ST96XSDValidator to analyze the name of the component works by first splitting the component names according to the camel casing. For instance, the component "MarkMultimediaFileFormatCategory" will be analyzed by splitting the terms: "Mark", "Multimedia", "File", "Format", and "Category". However, acronyms and abbreviations which appear next to each other will fail.For instance:

Common/**IPOASIS**TableType.xsd

Common/**WIPOST3**CodeType.xsd

In the examples above, both "IP" and "OASIS" are both approved acronyms, but the Validator is unable to determine that the component name is composed of both acronyms.

To remove these XSDs from the violations list for GD-14, add an entry in the ApprovedExceptions.xml file in the ST96XSDValidator/dev installation directory as follows:

<ApprovedExceptions>

...  
 <ApprovedException ruleID="GD-14" subdir="Common" file="ExtendedWIPOST3CodeType.xsd"/>

<ApprovedException ruleID="GD-14" subdir="Common" file="IPOASISEntryType.xsd"/>

<ApprovedException ruleID="GD-14" subdir="Common" file="IPOASISTable.xsd"/>

<ApprovedException ruleID="GD-14" subdir="Common" file="IPOASISTableType.xsd"/>

<ApprovedException ruleID="GD-14" subdir="Common" file="WIPOST3CodeType.xsd"/>

...  
</ApprovedExceptions>

These entries instruct the ST96XSDValidator to suppress violation reports of GD-14 for the listed XSDs.

If ApprovedException/@subdir is omitted, an XSD named ApprovedException/@file found in any subdir will be considered to be an approved exception.

### Approved Exceptions Update via Excel

For convenience, the ApprovedExceptions.xml file may be updated through Excel by adding or deleting rows as needed, and exporting the updates to the XML file.

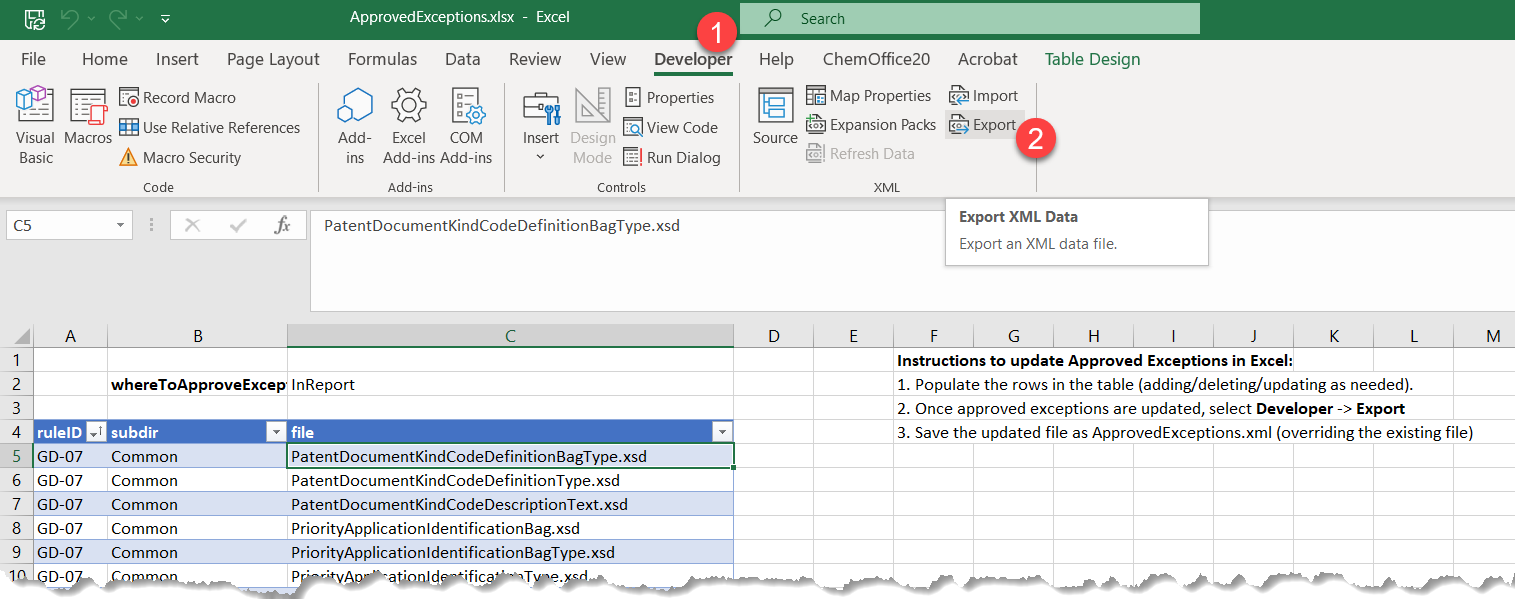


Figure 1: Updating the Approved Exceptions via Excel

## Spell Check Configuration

This section describes *optional* settings for indicating custom words to be used in the spell check reported in the Integrity Check. These should consist of IP-specific terminology that are not included in the dictionary used. The spell check will use the provided dictionary and affix files en\_GB\_oxendict.dic and en\_GB\_oxendict.aff (respectively). In addition, the spell check will automatically incorporate the terms in AcronymsAndAbbreviations.xml and AcronymsAndAbbreviationsLocal.xml to the custom words file.

Add IP-specific terminology not recognized by the Spell Check dictionary to the file CustomWords-ST96.txt. Each term should be on a separate line. There is no need to update the word count in the first line.

To add IP-specific terminology that might appear with different word stems, add a line to the CustomWords-ST96-with-stemming.txt file that is colon-delimited in the following format:

<new word>:<existing dictionary word that has the same prefix/stemming>

Example: “grantor:assessor”

Spell check usage: ST96SpellChecker [mandatory] 1 2 3 [optional] -a 4 -c 5 -o 6

mandatory arguments:

1. words file path=<path> path to file containing words to be checked, e.g st96words.txt

2. dictionary file path=<path> path to language dictionary file, e.g. en\_GB-large.dic

3. affix file path=<path> path to language affix file, e.g. en\_GB-large.aff

optional:

-a 4. words affix file path=<path> path to custom affix file containing words to be checked, e.g CustomWords-ST96-with-stemming.txt

-c 5. custom words file path=<path> path to custom words file, e.g. CustomWords-ST96.txt

-o 6. output xml file path=<path> path to output file, e.g. output.xml

-help, -h, -? print this message

Spell check is performed on words delimited by line breaks or commas.

Component Names:

* are split into words based on camel casing.

Enumeration Values:

* Assumptions: underscore is used in some cases where punctuation would normally be used (e.g. "As\_of\_right extension granted"); dashes were stripped (e.g. "Pre-Registration publication" became enumeration value "PreRegistration publication")
* values are split into words based on camel casing, underscore and space delimiters.
* Ignored: alphanumeric codes such as 2- or 3-letter codes (ST.3 codes, Key Event Codes in Legal Status, etc.)

Component Documentation:

* Ignored: hyperlinks, words composed of numbers, and ST96 components referenced with prefix (e.g. "com:FigureReference")
* Preprocessing to replace or remove punctuation.
* Documentation is split based on the following delimiters: space, comma (","), semicolon (";"), slash ("/") before performing spell check

Enumeration Value Documentation:

* Preprocessing to replace or remove punctuation.

Current limitations:

* The feature requires the input be tokenized into individual words prior to checking the spelling. This means that the spell check cannot properly check the spelling of compound words.

# Running ‘ST96XSDValidator’

## From the command line

The preferred way to run ST96XSDValidator is from the command line because it

produces the highest quality output reports and can test validation via a unit test suite of over 350 automated assertions.

### Checking Schema(s) in a Directory

To check an entire directory of XSDs (and the XSDs in the subdirectories within, recursively), type the following command from the ST96XSDValidator/dev installation directory:

[Windows:]

ST96XSDValidator "***path\_to\_input\_dir***"

[Mac OS or Linux:]

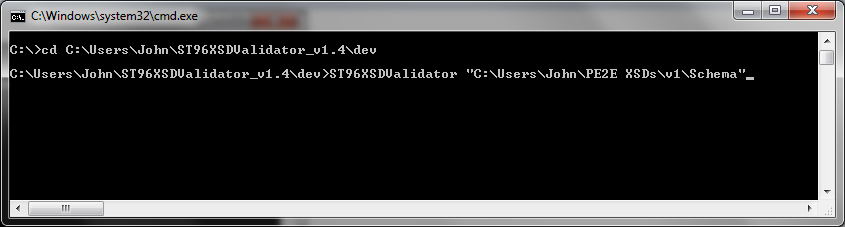
./ST96XSDValidator.sh "***path\_to\_input\_dir***"

For example, if the path to the input directory containing the XSDs to check is C:\Users\John\ST96XMLSchema\_V2\_1, and executable installation directory is c:\Users\John\ST96XSDValidator\_v1.1, issue the following command:

[Windows:]

cd C:\Users\John\ST96XSDValidator\_v1.4\dev

ST96XSDValidator "C:\Users\John\PE2E XSDs\v1\Schema"



[Mac OS or Linux:]

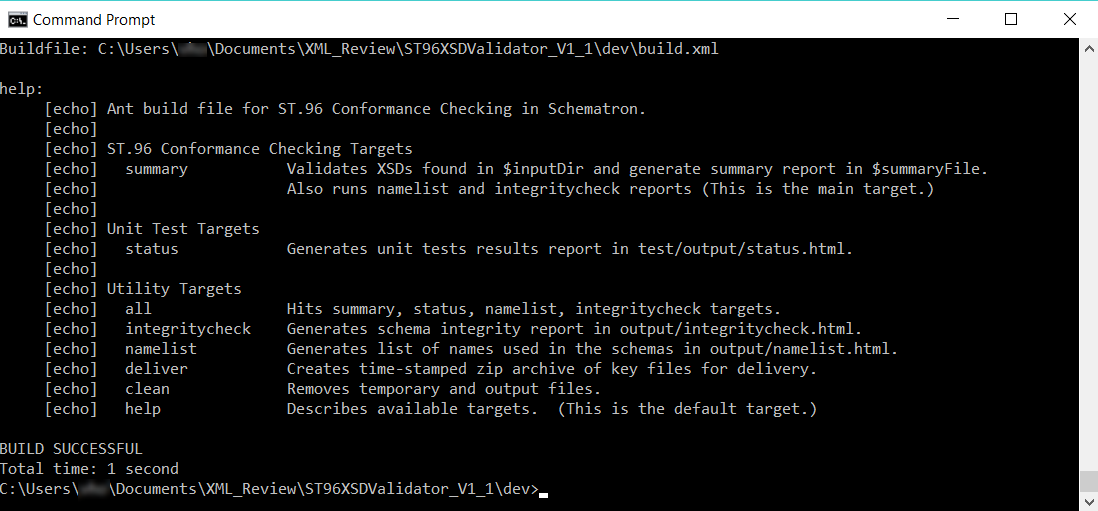
./ST96XSDValidator.sh "Users/John/PE2E XSDs/v1/Schema"

The results of the rules checking will be available in the output/summary.html file (as well as in a timestamped copy output/summary\_DATE\_TIME.html), an excerpt from which is shown below:

[Windows:]

ST96XSDValidator help

Will return a list of available targets:



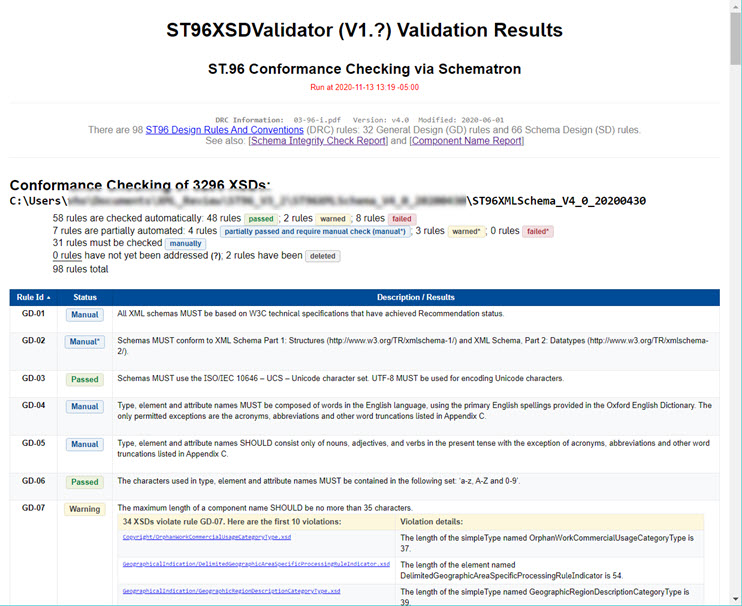


Figure 2: Validation Results Report

### Checking Schema(s) in a Directory, by comparing against a previous version of the Schema(s)

It is recommended to provide the directory of the previous version of the XSDs (and the XSDs in the subdirectories within, recursively), so that the ST96XSDValidator can identify new and modified components, and sort the results of any rule violations for these components first:

[Windows:]

ST96XSDValidator "***path\_to\_input\_dir***" "***path\_to\_previous\_schema\_dir***"

[Mac OS or Linux:]

./ST96XSDValidator.sh "***path\_to\_input\_dir***" "***path\_to\_previous\_schema\_dir***"

The results of new or modified components will precede unchanged components.

The results of the rules checking after providing the directory of the previous version of the schema is shown below:

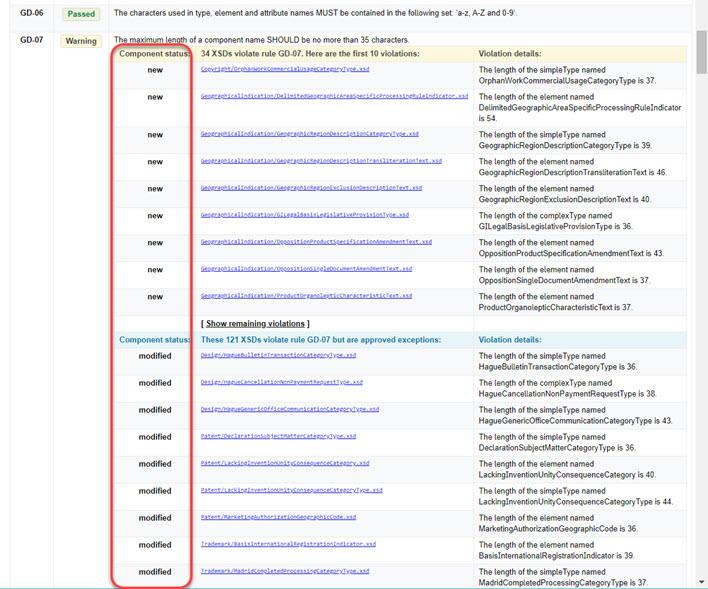


Figure 3: Validation Results Report

|  |
| --- |
| **Technical Note**: Developers familiar with Apache ant can bypass the xsdvalidate script and directly hit the ant target for generating a validation summary (summary). A list of other targets are available. Hit the “help” for details. |

## With a Graphical User Interface:

[Windows:]

ST96XSDValidator

[Mac OS or Linux:]

./ST96XSDValidator.sh

Double-clicking or calling the Validator with no arguments will launch a dialog asking the user to select the directory containing the XSD files, and optionally, the directory containing the reference XSD files (e.g. the previous version of the Schema), used for comparison.

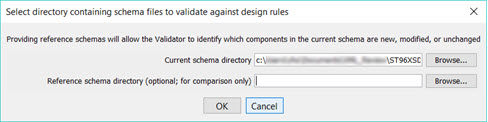


Figure 4: User Interface for XSD Validator

## Integrity checks

Integrity checks attempt to identify potential issues which are not directly related to a design rule, or which cannot be automated using Schematron. The Schema Integrity Check includes spell checking of component names or enumeration values, and checking of formalities such as missing data types, unused prefixes.

## Component Name Report

The Component Name report provides a list of Elements, Complex Types, Simple Types, and Attributes for each namespace.

If the previous schema directory is provided to the Validator, the report will indicate which components are identified as New or Modified, including a feature to toggle the components displayed according to revision status, as shown below:

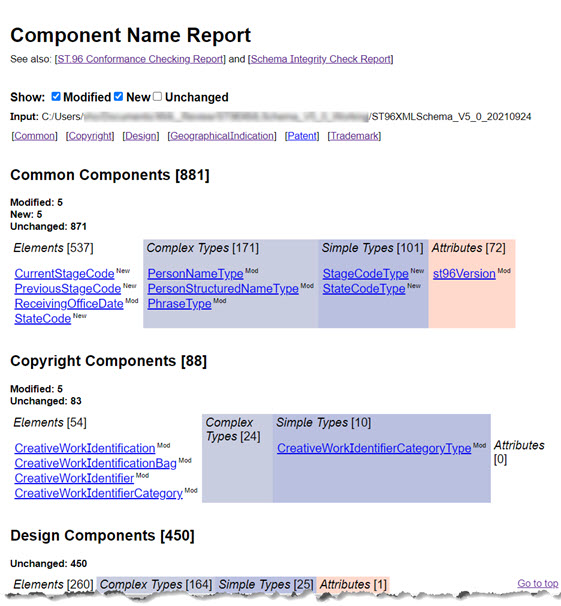


Figure 5: Component Name Report

## Referenced Component Lookup

The Referenced Component Lookup provides a lookup of referenced components in each namespace, and the corresponding components which use them.

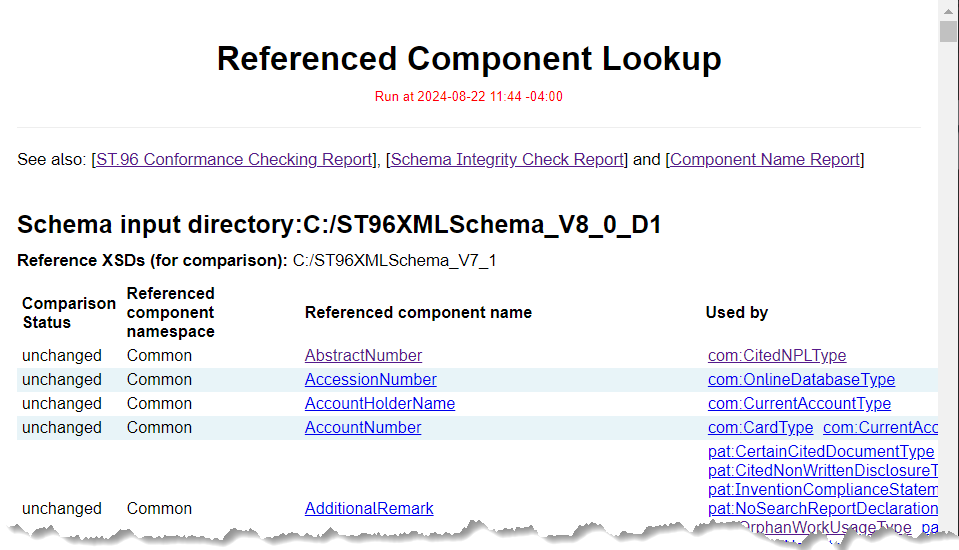


Figure 6: Referenced Component Lookup

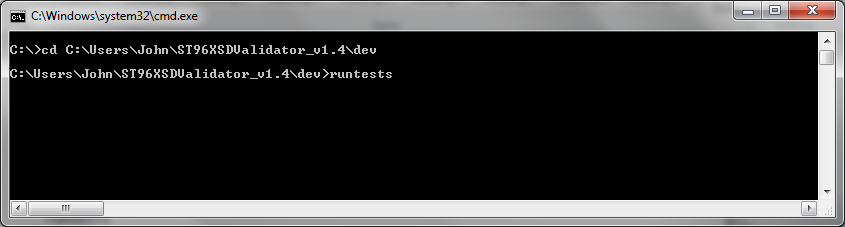
## Unit Tests

|  |
| --- |
| **Note**: Unit tests are used by developers during development and testing of xsdvalidate. Most of the time, regular users will want to use the xsdvalidate command as described in Section 1.9.1. |

To run the unit tests, type the following command from the ST96XSDValidator/dev directory:

[Windows:]

runtests



[Mac OS or Linux:]

./runtests

The results of the unit tests will be available in the output/status.html file, an excerpt from which is show below:

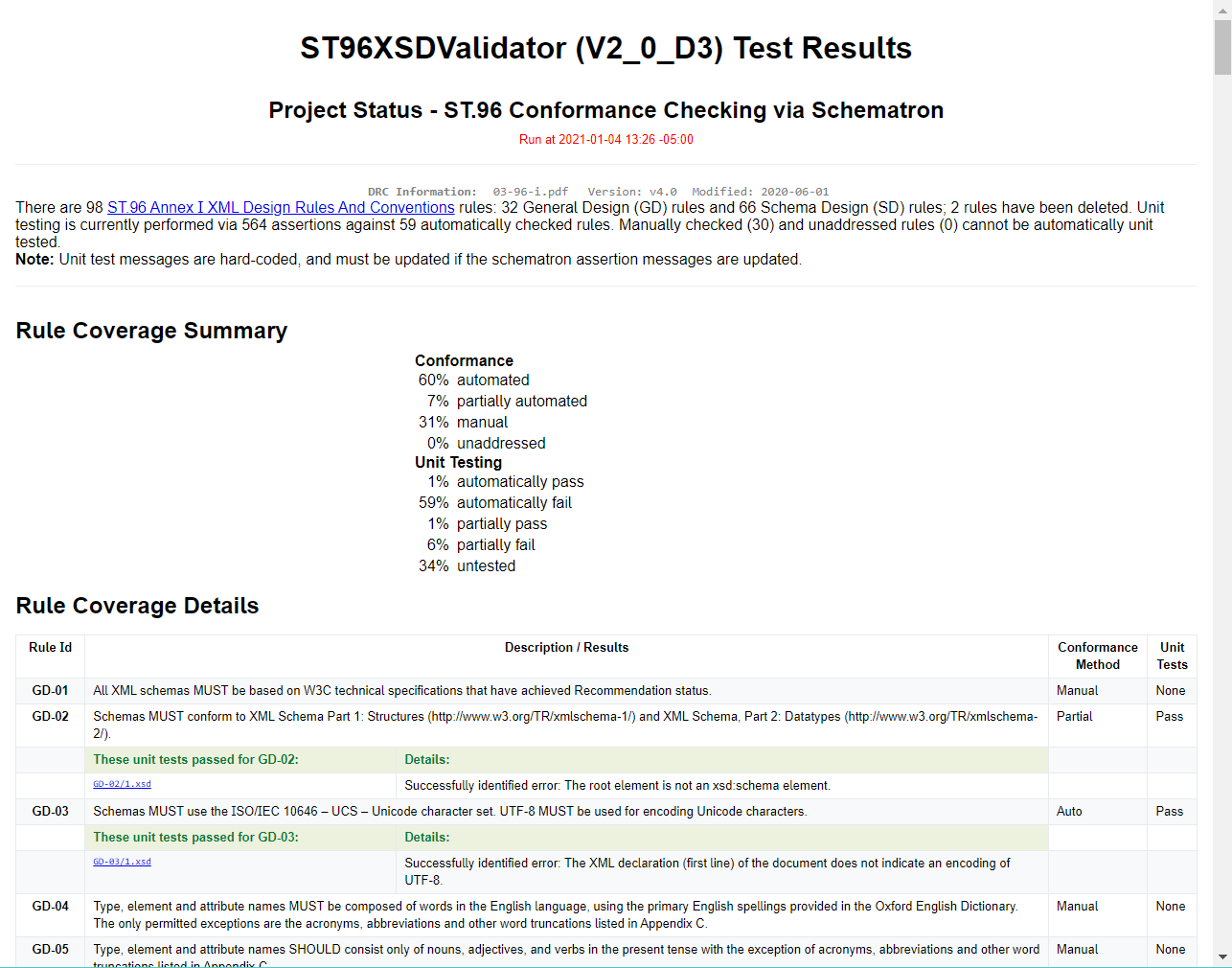


Figure 7: Project Status Report

Further details regarding unit testing of ST96XSDValidator can be found in Appendix B: Unit Testing Framework.

## Using oXygen XML Editor

The oXygen XML Editor can validate a document using a Schematron schema[[1]](#footnote-2). This capability can be used in a limited manner to validate an XSD against the Schematron schema used in ST96XSDValidator.

|  |
| --- |
| **Note**: The preferred way to run ST96XSDValidator is via the command line. Unlike the reports produced via ant build or Windows batch file, error and warning messages in oXygen show neither ST.96 rule identifier nor rule text. See **Section 1.9** for further advantages of running ST96XSDValidator via the command line. |

### Checking a Single File

To check a single file using The oXygen XML Editor:

1. Open the XSD to be validated.
2. Choose the menu item **Document**/**Validate**/**Validate with...**
3. Choose the tab labeled **Schematron Schema**.
4. In the URL box, navigate to the location of the Schematron schema (ST96XSDValidator/dev/schemas/wipo\_xsd.sch).
5. Click OK.

The resulting error messages will appear in a separate results box, as shown in below. For many error messages, clicking on the error message will show the line in the document that triggered that message.

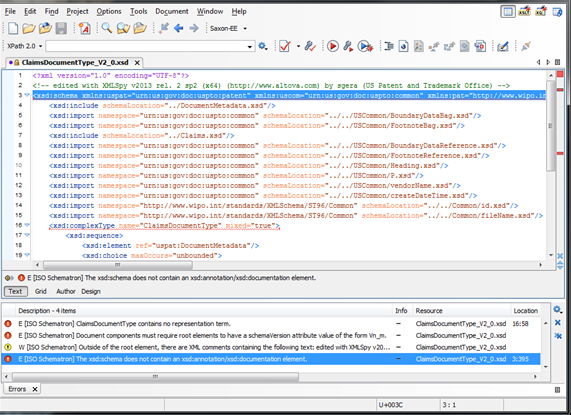


Figure 8: Errors/Warnings in the oXygen XML Editor

# References

* WIPO Standard ST.96 Annex I - XML Design Rules and Conventions (version 8.0)
* Extensible Markup Language (XML) 1.0, Second Edition, Tim Bray et al., eds., W3C, 6 October 2000.
* XML Schema Part 1: Structures, Henry S. Thompson et al., eds, W3C, 28 October 2004.
* ISO Schematron reference implementation, (iso-schematron-xslt2.zip, for XSLT 2.0).
* Schematron & SVRL Cheat Sheet, Rick Jelliffe

# Appendix A: Rules Enforced by the Schematron

Table 1 lists the rules from the *XML Standards and Web Services using XML* document and indicates which ones are enforced automatically.

**Note**: Exceptions to automated findings should not be allowed for any Rules with MUST/MUST NOT clauses, unless there is a possibility of false positives due to limitations of automation.

Table 1: Enforceability of XML rules

| **ST. 96 rule** |  | **Automation concern** | **Assumption to automate** | **Exceptions allowed?** |
| --- | --- | --- | --- | --- |
| GD-01 | M | XSDs must use http://www.w3.org/2001/XMLSchema namespace anyway. | Not applicable. | N/A |
| GD-02 | M\* | Validation against all of XSD 1.0 spec is very complicated. | Not applicable.  Assumption is that tools used to develop schemas (e.g., Altova XMLSpy or oXygen XML Editor) will conform to W3C recommendations. | N/A |
| GD-03 | A | Schematron and XSLT has no provision for querying encoding. | Can check the XML declaration for the encoding. | No |
| GD-04 | M | Referencing Oxford English Dictionary is difficult. | Not applicable. Include a link to the Integrity check (which now implements spell checking) instead. | N/A |
| GD-05 | M | Knowing parts of speech and tense is difficult. | Not applicable. Include a link to the Integrity check (which now implements spell checking) instead. | N/A |
| GD-06 | A | No concerns. | No assumptions required to automate. | No |
| GD-07 | A | No concerns.  overlaps with GD-12, GD-13, GD-14 | Exclude ‘Type’ from complexType or simpleType string length constraint, since GD-10 has requirement for the suffix ‘Type’, which in most cases is added to the name of the element/attribute which uses the type. | Yes |
| GD-08 | M | “Concise and self-explanatory” requires judgement. | Not applicable. | N/A |
| GD-09 | A | Referencing dictionary is difficult.  Knowing word boundaries is difficult. | Can detect these problems:   * All capitals. * Wrong initial letter capitalization.   Cannot detect these sorts of problems, for example:   * BadelementName * BadElementname   Assumption that these problems would be detected as misspellings in the Integrity check  Names less than 3 characters are not checked. | No |
| GD-10 | A | Referencing dictionary is difficult.  Knowing word boundaries is difficult. | Can detect these problems:   * All capitals. * Wrong initial letter capitalization.   Cannot detect this sort of problems, for example:   * BadnameType * Assumption that these problems would be detected as misspellings in the Integrity check | No |
| GD-11 | A | Referencing dictionary is difficult.  Knowing word boundaries is difficult. | Can detect these problems:   * All capitals. * Wrong initial letter capitalization.   Cannot detect this sort of problem, for example:   * bigbadAttribute * Assumption that these problems would be detected as misspellings in the Integrity check | No |
| GD-12 | A |  | Assumption that camel-casing is proper.  After splitting name by camel case, can assume that word corresponding to expanded Appendix C abbreviation should be replace by corresponding abbreviation.  Will not be able to find consecutive acronyms/abbreviations. | Yes |
| GD-13 | A |  | Assumption that camel-casing is proper.  After splitting name by camel case, can compare words which are all uppercase against Appendix C abbreviations.  Can determine if the first word corresponding to an Appendix C abbreviation appears with the proper capitalization. | Yes |
| GD-14 | A | Cannot detect all variations of an abbreviation. | Assumption that camel-casing is proper.  After splitting name by camel case, can compare words which are all uppercase against Appendix C abbreviations.  Can check that approved abbreviations/acronyms have appropriate case.  Can detect this sort of problem, for example:   * SerialALTTitleText where "ALT" does not have the same casing as appendix C "Alt"   Cannot verify against Abbreviations which use UCC, for example:   * BioDeposit (since the component name would have been split to "Bio" and "Deposit" first, before comparing against Appendix C).   Will not be able to find consecutive acronyms/abbreviations. | Yes |
| GD-15 | M\* | “Meaningful” is subjective and requires human judgement (“meaningful Object Class Term”). | Can only identify if there is no Object class term (i.e. there is a word in the name that is not “Type” or a representation term). | Yes |
| GD-16 | M | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically.  Names of basic elements (those based on a simple or complex type that do not permit children) should include a Representation Term “where applicable.” | Not applicable. | N/A |
| GD-17 | M\* | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically. | Can only identify if there is no Object class term and/or Property term. In other words, can identify if there are not at least two words in the name. | Yes |
| GD-18 | M | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically. | Not applicable. | N/A |
| GD-19 | M | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically. | Not applicable. | N/A |
| GD-20 | M | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically. | Not applicable. | N/A |
| GD-21 | M | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically. | Not applicable. | N/A |
| GD-22 | A | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically.  Cannot identify “equivalent word.” | Assume duplicated Representation Term is a sign of violation of this rule. | No |
| GD-23 | A | Meaning of parts of names (Object Class, Property Term, Representation Term, Qualifier Term per ISO 11179 part 5) cannot be assessed automatically.  Per GD-17, Names of basic elements (those based on a simple or complex type that do not permit children) should include a Representation Term “where applicable.”  This rule applies when a representation term is required. | Assume presence of any word listed in Appendix B Representation Terms satisfies this rule.  Exclude 'Code' from checking if a different representation term should be used, since SD-38 already covers this, and it is a manual/semi-manual check.  Use Appendix B Data Types.  Since xsd:token is a subset of xsd:string, it can also be used for the representation terms "Text" and "Name"  A “Basic component” is an xsd:attribute or an xsd:element with data type of: com:AmountType, xsd:token, com:DateType, xsd:date, xsd:boolean, com:MeasureType, xsd:string, xsd:positiveInteger, xsd:decimal, com:QuantityType, xsd:nonNegativeInteger, xsd:time, xsd:dateTime, xsd:anyURI, com:IdentifierType, com:LocalizedTextType, com:OrderedTextType, com:PhraseType, or having a xsd:simpleType or xsd:simpleContent definition, or other xsd data type.  Assume that a basic component must have a representation term. | Yes |
| GD-24 | A | No concerns. | Use collection() to check @name and @targetNamespace for duplicates.  For performance reasons, pre-check GD-24 separately since wipo\_xsd.sch is applied to each design schema, but as a fallback, can check GD-24 for duplicates inline. | No |
| GD-25 | M\* | Plurality formation is too irregular to automate without dictionary access.  Can only do a partial pattern match for the most common plural form using "s" | Not applicable.  Explicitly ignore ‘GoodsServices’ and ‘Claims’ in component names, as these are business terms referenced in the plural (they will not trigger an error requiring an exception). | Yes |
| GD-26 | M\* | No concerns. | Assume a "collection" can be:  - xsd:choice with @maxOccurs = 'unbounded'  - xsd:sequence with xsd:element @maxOccurs = 'unbounded'  - xsd:choice with xsd:element @maxOccurs = 'unbounded'  - complexTypes extended from other types and including the above | Yes |
| GD-27 | M\* | “Unless part of the business terminology” cannot be assessed automatically. | Not applicable.  Connecting words are indicated in file ConnectingWords.xml | Yes |
| GD-28 | M | “Translated, changed or replaced” cannot be assessed automatically. | Not applicable. | N/A |
| GD-29 | M\* | Detecting presence of ‘article’ or ‘rule’ could cause false positive errors. | Legal references are specified in file LegalReferencePrefixes.xml  Could accept any false positive errors. | Yes |
| GD-30 | A | No concerns. | No assumptions required to automate. | No |
| GD-31 | A | Cannot be sure whether a file is intended to be a draft (GD-32).  Document-level schemas cannot be identified based on the schema content.  Component name cannot be determined from flattened schemas. | Assume if “draft” or “D[0-9]” appears in file name, it is intended to be a draft; otherwise, assume it is not intended to be a draft.  Document-level schemas can only be identified if they are in a folder named “Document.”  Does not apply to flattened schemas.  Assume @version attribute must match filename when version information is present in the document-level schema filename. | No |
| GD-32 | A | Cannot be sure whether draft is based on existing schema or is new.  Document-level schemas cannot be identified based on the schema content.  Component name cannot be determined from flattened schemas. | Assume draft is based on existing schema.  Document-level schemas can only be identified if they are in a folder named “Document.”  Does not apply to flattened schemas. | Yes |
| GD-33 | A | Can only determine a schema is a test schema if filename has ‘\_T’ or ‘test’  Document-level schemas cannot be identified based on the schema content.  Component name cannot be determined from flattened schemas. | Document-level schemas can only be identified if they are in a folder named “Document.”  Does not apply to flattened schemas. | Yes |
| SD-01 | A | The xsd:import construct must be used to bring in files from other namespaces, so this rule appears to constrain nothing beyond that which is already constrained by XML Schema itself.  Also, if a Patent, Trademark, or Design XSD does not require anything from Common, why must it import Common? | Assume XSDs are valid W3C XML Schemas.  Since an XSD is valid if xsd:import is empty, check to ensure @schemaLocation is utilized (@namespace is implied as the referenced component must have a targetNamespace).  Since an XSD will be invalid if the targetNamespace of the referenced schema in an xsd:include does not match the targetNamespace of the current schema, there is no need to check xsd:include.  If the common namespace prefix is declared on xsd:schema, and there is an xsd:import, activate this rule. | No |
| SD-02 | A | No concerns. | Assume improper reference to Patent/Trademark/Design/GI/Copyright if a reference/import is made to a namespace which is not the Common namespace or an external namespace (OASIS Table and MathML) | No |
| SD-03 | A | No concerns. | No assumptions required to automate. | No |
| SD-04 | M\* | “Wherever applicable” cannot be assessed automatically. | Assume complexType or simpleType models in the same namespace should be unique, as the type would normally be reused instead of creating a new type.  For performance reasons, pre-check SD-04 separately since wipo\_xsd.sch is applied to each design schema, but as a fallback, can check SD-04 for duplicates inline. | Yes |
| SD-05 | M\* | “Maximum extent possible” cannot be assessed automatically. | Assume Common components can be used instead of creating component in a different namespace (with the same name and/or model).  For performance reasons, pre-check SD-05 separately since wipo\_xsd.sch is applied to each design schema, but as a fallback, can check SD-05 inline. | Yes |
| SD-06 | A | No concerns. | No assumptions required to automate. | No |
| SD-07 | A | No concerns. | No assumptions required to automate. | No |
| SD-08 | A | No concerns. | No assumptions required to automate. | No |
| SD-09 | M\* | Cannot detect changes to published namespaces. | Assume "change" if namespace does not correspond to one of the namespaces specified in SD-15 (indicated in file BaselineNamespaces.xml). | Yes |
| SD-10 | A | No concerns. | No assumptions required to automate. | No |
| SD-11 | A | Namespace qualifications[1] must be used to reference XML Schema constructs anyway, so this rule appears to constrain nothing beyond that which is already constrained by XML Schema itself.  [1] Definition: A qualified name is a name subject to namespace interpretation [...] Syntactically, they are either prefixed names or unprefixed names.  SD-15 already says that “Schemas SHOULD use “xsd” as a namespace prefix” | Check that namespace *prefixes* are used to reference XML Schema constructs.  Where SD-15 says the “xsd” prefix SHOULD be used, here just say that some namespace prefix MUST be used. | No |
| SD-12 | A | There are many schemes and checking for the URI syntax format would require robust regular expression. | Assume URI-formatted namespace if starts with one of the following:  - "http:"  - "https:"  - "urn:"  Assume not URI-formatted if results of fn:iri-to-uri is a different string from input | No |
| SD-13 | A | No concerns. | No assumptions required to automate. | No |
| SD-14 | A | No concerns. | No assumptions required to automate.  Applies to WIPO, not IP-specific, XSDs. | No |
| SD-15 | A | SD-15 says "xsd" prefix SHOULD be used, but all other rules presume its use (SD-07, SD-13, SD-16, SD-53, SD-54, SD-56, SD-57, SD-58, SD-62), some of which have "MUST" assertions. | No assumptions required to automate. | Yes |
| SD-16 | A | No concerns. | No assumptions required to automate. | No |
| SD-17 | A | No concerns. | No assumptions required to automate. | No |
| SD-18 | A | No concerns. | Common components are found in a “Common” subdirectory.  Applies to WIPO, not IP-specific, XSDs. | No |
| SD-19 | A | No concerns. | Patent components are found in a “Patent” subdirectory.  Applies to WIPO, not IP-specific, XSDs. | No |
| SD-20 | A | No concerns. | Trademark components are found in a “Trademark” subdirectory.  Applies to WIPO, not IP-specific, XSDs. | No |
| SD-21 | A | No concerns. | Design components are found in a “Design” subdirectory.  Applies to WIPO, not IP-specific, XSDs. | No |
| SD-22 | A | No concerns. | No assumptions required to automate. | Yes |
| SD-23 | M | Cannot detect validation across versions of an XSD. | Not applicable. | N/A |
| SD-24 | M | Cannot detect validation across versions of an XSD. | Not applicable. | N/A |
| ~~SD-25~~ | D | *Deleted* |  |  |
| ~~SD-26~~ | D | *Deleted* |  |  |
| SD-27 | M | Cannot track version changes of XSDs. | Provide access to past versions of XSDs. | N/A |
| SD-28 | A | Cannot track version changes of XSDs. | Provide access to past versions of XSDs.  Assume st96Version value is authoritative version that all schemas in release must correspond to. | No |
| SD-29 | M | Cannot track version changes of XSDs. | Provide access to past versions of XSDs. | N/A |
| SD-30 | A | No concerns. | No assumptions required to automate. | No |
| SD-31 | A | Cannot determine root element in flattened schemas. | In order to determine root element, must be either design schema, or have only a single xsd:element component. | No |
| SD-32 | M | Cardinality and granularity conformity across three past standards (ST.36, ST.66 or ST.86) is difficult to assess. | Not applicable. | N/A |
| SD-33 | A | “Unless they are needed” is difficult to assess automatically. | Detect “Other” or “Undefined” without judging how needed they might be. | Yes |
| SD-34 | A | Math formula are difficult to detect automatically.  Type usage is difficult to ascertain because of the complexities of type and composition definition in XSD.  MathML namespace (http://www.w3.org/1998/Math/MathML) should normally not change, and therefore it should not be possible to ascertain which version of MathML is utilized.  However, MathML3 was implemented with a different namespace: http://www.w3.org/1998/Math/MathML3 | If an element name contains ‘math’ in any place in any case, conclude that it contains a math formula.  Covering common XSD constructs and limiting recursive search to a depth of 6 is sufficient for a uses-type() function to answer correctly  in most cases.  As a result of namespace implementation for past schemas, this rule should fail for XSDs based on ST.96 V7\_1 and prior versions. | Yes |
| SD-35 | A | Tables are difficult to detect automatically.  Type usage is difficult to ascertain because of the complexities of type and composition definition in XSD. | If an element name contains ‘table’ in any place in any case, and the element name does not appear in the following list, conclude that the element contains a table: tablebody, tablegroup, tableimage, tabletitle, tabledata, tablecell, tabledatacell, tablefooter, tableheader, tableheadercell, or tablerow.  Covering common XSD constructs and limiting recursive search to a depth of 6 is sufficient for a uses-type() function to answer correctly  in most cases. | Yes |
| SD-36 | A | There is no published list of approved industry-standard schemas. | Limit scope to ST.96 and assume that only OASIS Tables and MathML are approved to be imported.  Allow both MathML namespaces:  http://www.w3.org/1998/Math/MathML3 (used in ST.96 V7\_1 and prior versions), and http://www.w3.org/1998/Math/MathML (used from ST.96 V8\_0 and beyond) | Yes |
| SD-37 | M | “Maximum extent possible” is difficult to assess. | Define an acceptable ratio of simple types to complex types.  Could partially check by making sure that the major simple types (Language Code, two Country Codes) are used. | N/A |
| SD-38 | M\* | What is it that is not allowed?  overlaps with SD-39 | As long as some (nested) component has simpleType/restriction/enumeration | Yes |
| SD-39 | M | What is it that is not allowed?  overlaps with SD-38 |  | N/A |
| SD-40 | M\* | Which elements are “used for representing IPOs and for priority and designated country/organization”?  Must be able to distinguish when an IPO is being referenced vs a country (for residence/address purposes), since ST3 Codes are a superset of ISO country codes.  overlaps with SD-41 | Assume element name contains ‘officecode’, or ‘contractingparty’ or ‘authoritycode’.  Check that com:WIPOST3CodeType is used explicitly, or that every IP Office code enumeration value is listed in com:WIPOST3CodeType or com:ExtendedWIPOST3CodeType  Assume name may not contain ‘residence’ or ‘domicile’ | Yes |
| SD-41 | M\* | Which elements are “used for the representation of the names of countries, dependencies, and other areas of particular geopolitical interest”?  Must be able to distinguish when an IPO is being referenced vs a country (for residence/address purposes), since ST3 Codes are a superset of ISO country codes.  overlaps with SD-40 | Assume element name contains ‘country’, but may not contain ‘designated’ or ‘priority’.  Check against codes listed in com:ISOCountryCodeType or com:Extended ISOCodeType | Yes |
| SD-42 | A | Which elements contain language codes? | Assume element name contains ‘language’.  Check against com:ISOLanguageCodeType or com:ExtendedISOLanguageCodeType | Yes |
| SD-43 | A | Which elements contains date and time data? | Overlaps with GD-23, which includes checks that the representation terms Date/Time/DateTime are used appropriately.  Assume element/attribute name containing ‘date’ or ‘time’ must use schema data or time data types, unless the representation term is ‘Category’ | Yes |
| SD-44 | A | Which elements contains currency data? | Assume element name contains ‘currency’ or ‘money’. | Yes |
| SD-45 | A | No concerns. | No assumptions required to automate. | No |
| SD-46 | M | Judgement required. | Not applicable.  Include enumeration values for manual inspection.  Exclude ISO codes and those enumeration values which appear to correspond to alphanumeric codes. | N/A |
| SD-47 | M | What is it that is not allowed? | List any abstract complexTypes for review. | N/A |
| SD-48 | M | Difficult to automatically determine “non-business data”. | Not applicable.  Include attribute annotation documentation for manual inspection. | N/A |
| SD-49 | A | No concerns. | No assumptions required to automate. | Yes |
| SD-50 | A | No concerns. | No assumptions required to automate. | No |
| SD-51 | A | No concerns. | No assumptions required to automate. | Yes |
| SD-52 | A | No concerns. | No assumptions required to automate. | Yes |
| SD-53 | M | What is it that is not allowed? | Just check if maxOccurs is less than the number of components (and conversely if minOccurs is greater than the number of components) in the xsd:sequence or xsd:choice. | N/A |
| SD-54 | A | Cannot assess intent (“for accessibility”). | Just check that xsd:any is not used. | No |
| SD-55 | A | No concerns. | No assumptions required to automate. | No |
| SD-56 | M | Cannot assess “appropriate” automatically (“Schemas SHOULD use xsd:key​/xsd:keyref​/xsd:unique and​/or xsd:ID​/xsd:IDREF​/xsd:IDREFS to identify constraints as appropriate”). |  | N/A |
| SD-57 | M | Cannot assess “appropriate” automatically (“Schemas SHOULD use xsd:ID​/xsd:IDREF​/xsd:IDREFS for a single reference or multiple references within the current XML document to identify constraints as appropriate”). | Just list components that use xsd:IDREFS, xsd:IDREF, com:idref, or com:idrefs for manual review | N/A |
| SD-58 | M\* | Cannot interpret documentation (“SHOULD NOT contain implementation details or other information not directly related to themeaning of the construct”). | Just detect presence of xsd:documentation on element and attribute definitions. | Yes |
| SD-59 | A | No concerns. | No assumptions required to automate. | Yes |
| SD-60 | M\* | Cannot interpret documentation.  Can do extremely limited pattern match on documentation in instances where enumerations are not used. | Looks for the term "value" in the documentation. Assume that mention of the term "value" could refer to possible values or the form of the expected value.  There will likely be false positives. | Yes |
| SD-61 | A | No concerns. | No assumptions required to automate. | No for mandatory documentation |
| SD-62 | M | Cannot assess scope requirements (“Schemas SHOULD use xsd:key​/xsd:unique​/xsd:keyref within a scope where uniqueness has to stay entirely within that scope”). | Just list components that use xsd:key/xsd:unique/xsd:keyref for manual review. | N/A |
| SD-63 | A | Cannot determine root element in flattened schemas. | In order to determine root element, must be either design schema, or have only a single xsd:element component.  All valid ST.3 codes are accepted.  Can only check value of ipoVersion for IPO-specific schemas. | No |
| SD-64 | A | Flattened schemas are assumed to contain more than one component definition within the file.  Document-level schemas cannot be identified based on the schema content.  GD-31 already accounts for document-level schema filename having the correct format | Document-level schemas can only be identified if they are in a folder named “Document.”  In the ST.96 schema release folder structure, the release version number is followed by component types, i.e., Common, Copyright, GeographicalIndication, Patent, Trademark, Design and External Standards. For example, the version 3.0 of Contact component is placed in the folder, ST96/V3\_0/Common.  Assume folder containing namespace folders is the schema release folder. | No |
| SD-65 | A | No concerns. | Geographical Indication components are found in a “GeographicalIndication” subdirectory.  Applies to WIPO XSDs and those IP Offices which are responsible for geographical indication protection. | No |
| SD-66 | A | No concerns. | Copyright components are found in a “Copyright” subdirectory.  Applies to WIPO, not US-specific, XSDs. | No |
| SD-67 | M | Cannot detect validation across versions of an XSD. | Not applicable. | N/A |

Table 1: M = Manual check; M\* = Partial manual check; A = Auto check

# Appendix B: Unit Testing Framework Details

Each unit test is written specifically to a targeted, automatically checked, ST.96 rule. Each unit test is applied in isolation of all ST.96 rules other than the targeted rule. If the entire ST.96 Schematron file (ST96XSDValidator/dev/schemas/wipo\_xsd.sch) were to be used *as-is* in unit testing, there could easily be cross-rule violations that would distract from the targeted test results or unnecessarily complicate creation of the targeted test. Instead, wipo\_xsd.sch is automatically split into separate, self-contained Schema files, and the unit tests are applied to these separate Schema files in isolation.

The following subdirectories exist under ST96XSDValidator/dev/test in order to support unit testing of ST96XSDValidator on a per-ST.96-rule basis:

* **input/**

Contains one subdirectory of XSD test cases for each ST.96 rule.

* **output/**

Contains status.html, the combined and formatted results of all unit tests.

* **schemas/split\_guidelines**

Contains separate (automatically split) Schematron files for each ST.96 rule found in ST96XSDValidator/dev/schemas/wipo\_xsd.sch.

* **schemas/split\_guidelines\_test**

Contains a hand-written Schematron file for each automatically checked ST.96 rule. (Unit tests for ST96XSDValidator are themselves implemented in Schematron.)

* **svrl\_conformance/**

Contains results of applying the separated ST.96 Schematron files to each unit test XSD.

* **svrl\_conformance\_svrl/**

Contains results of applying the hand-written unit test Schematron files (from schemas/split\_guidelines\_test) to the results (in svrl\_conformance) of applying the separated ST.96 Schematron tests to the unit test input XSDs.

* **xsl/status.xsl**

Hand-written XSLT file that formats results of unit testing (output/status.html).

* **xsl/split.xsl**

Hand-written XSLT file that splits ST.96 Schematron into separate files per rule.

* **xsl/split\_guidelines**

Contains XSLT compiled from schemas/split\_guidelines.

* **xsl/split\_guidelines\_test**

Contains XSLT compiled from schemas/split\_guidelines\_test, and test\_common.xsl (which contains the functions used for checking whether a unit test passed or failed)

For instance, in order to create a new unit test for a design rule, add a folder in test/input corresponding to the ID of the design rule.

* Add a test input XSD file to this new directory:
  + e.g. test/input/<rule ID>/<schema to test>.xsd
* Create a Schematron file in test/schemas/split\_guidelines\_tests corresponding to the ID of the design rule:
  + e.g. test/schemas/split\_guidelines\_tests/<rule ID>.sch
  + the rule context should be svrl:schematron-output
  + Use Schematron report to confirm that the conformance tests work as expected
* All input in the rule ID test subdirectory will be validated against the individual design rule. Place components which are expected to pass the rule in a separate file, to ensure that the test for the rule ID was triggered for that component.

Verify that the input passed the conformance test:

* create report for pass tests:

<let name="passtest1" value="fnx:expect-infile-by-component('Br', '1.svrl', $inFileName, /)"/>

<report test="$passtest1 != ''" flag="AUTO" diagnostics="uri" role="ERROR" see="#SD-50">

<value-of select="fnx:rule-test-outcome(true(), $passtest1)"/>

</report>

* create report for fail tests:

<let name="failtest3" value="fnx:expect-infile-by-component('f', '3.svrl', $inFileName, /)"/>

<report test="$failtest3 != ''" flag="AUTO" diagnostics="uri" role="ERROR" see="#SD-50">

<value-of select="fnx:rule-test-outcome(false(), $failtest3)"/>

</report>

|  |  |
| --- | --- |
| **Function** | **Returns** |
| fnx:expect-infile-by-component(xsd:string, xsd:string, xsd:string,node()) | * “Identified error for…” (the component/schema was tested and returned an error message) * “Failed to check component:…” (the test was not triggered because it did not match a rule context) * “No error for…” (the component/schema was tested and did not return an error message) * The second xsd:string corresponds to the svrl file name with the expected error message (i.e. the same name as the xsd test file, but with the extension svrl ) |
| fnx:rule-test-outcome(xsd:boolean, xsd:string) | * “Successfully validated:…” (pass test passed without error; fail test resulted in error) * “Failed:…” |

1. Note that only oXygen is discussed here because other major XML editors such as Altova XMLSpy do not currently have direct support for Schematron validation. However, note that a third-party, [XML Buddy](http://www.xml-buddy.com/schematron-validation-tool.htm), has a plug-in that provides [Schematron support](http://www.xml-buddy.com/schematron-validation-tool.htm) in XMLSpy. [↑](#footnote-ref-2)