



Guide to Schematron Rules and Patterns

Whitelist Schematron Guide

Version 2019-MAR

December 1, 2022

Distribution Notice:

This document has been approved for Public Release and is available for use without restriction.

Table of Contents

Chapter 1 - Introduction	1
1.1 - Purpose	1
1.2 - Overview	1
1.3 - Schematron	1
1.4 - Conformance	1
Chapter 2 - Rules	2
2.1 - ../Rules/WLAtomicEnergyMarkings_ID_00001.sch	3
2.2 - ../Rules/WLClassification_ID_00001.sch	4
2.3 - ../Rules/WLDisseminationControls_ID_00001.sch	5
2.4 - ../Rules/WLFGISourceOpen_ID_00001.sch	6
2.5 - ../Rules/WLFGISourceProtected_ID_00001.sch	7
2.6 - ../Rules/WLISMAAttributes_ID_00001.sch	8
2.7 - ../Rules/WLNtkAccessPolicy_ID_00001.sch	9
2.8 - ../Rules/WLNtkAccessPolicy_ID_00002.sch	10
2.9 - ../Rules/WLNtkAccessProfileValue_ID_00001.sch	11
2.10 - ../Rules/WLNtkAccessProfileValue_ID_00002.sch	12
2.11 - ../Rules/WLNtkAccessProfileValue_ID_00003.sch	13
2.12 - ../Rules/WLNtkVocabulary_ID_00001.sch	14
2.13 - ../Rules/WLNtkVocabulary_ID_00002.sch	15
2.14 - ../Rules/WLReleaseableTo_ID_00001.sch	16
2.15 - ../Rules/WLReleaseableTo_ID_00002.sch	17
2.16 - ../Rules/WLSARIdentifier_ID_00001.sch	18
2.17 - ../Rules/WLSCIcontrols_ID_00001.sch	19
2.18 - ../Rules/WLdisplayOnlyTo_ID_00001.sch	20
2.19 - ../Rules/WLnonICmarkings_ID_00001.sch	21
2.20 - ../Rules/WLnonUSControls_ID_00001.sch	22
2.21 - ../Rules/WLownerProducer_ID_00001.sch	23
2.22 - ../Rules/WLownerProducer_ID_00002.sch	24
Chapter 3 - Abstract Patterns	25
3.1 - ../Lib/AllValuesExistInList.sch	26
3.2 - ../Lib/SomeValuesExistInList.sch	27
3.3 - ../Lib/ValidateDecomposableTokensExistInWhitelist.sch	28
3.4 - ../Lib/ValidateTokenValuesNotInBlacklist.sch	29
3.5 - ../Lib/ValidateTokensExistInWhitelist.sch	30
Chapter 4 - Schematron Schema	31
4.1 - ../Whitelist_XML.sch	32
Chapter 5 - Removed Rules	39

Chapter 1 - Introduction

1.1 - Purpose

This is an informative supplement for Whitelist. This guide is generated from the Whitelist Schematron rules and provides a consolidated reference for the business rules of this specification.

1.2 - Overview

Chapter 2 is a listing of all the numbered rules in Whitelist. For each rule, there is a rule description, a code description, and a code block with the Schematron rule.

Chapter 3 is a listing of abstract patterns used in Whitelist. The abstract patterns may be used in numbered rules or provided as reference for use in rules developed by users of Whitelist. Each abstract pattern has a code description and a code block with the abstract Schematron pattern.

Chapter 4 is a listing of the master Whitelist Schematron file with all of the imports of rules and patterns. Many of the rules and patterns listed in Chapters 3 and 4 rely on functions and variables defined in the master file.

Chapter 5 is a listing of rules that have been deleted.

1.3 - Schematron

The business rules for Whitelist are encoded using ISO Schematron. Schematron is a rule-based validation language that uses XML Path Language to make assertions about an XML document.

Whitelist uses the XSLT 2.0 implementation of Schematron by Rick Jelliffe (2010-04-14) as its reference implementation. The only available identifying descriptors for this implementation are the implementer's name and date of release. This implementation may be found at the following URL: <http://code.google.com/p/schematron/>.



Important

The Schematron rules in this specification use XSLT 2.0 query binding.

1.4 - Conformance

This guide is informative. The Schematron rules listed here are normative in the sense that they convey criteria that a document **MUST** adhere to, exactly as English may be used to convey normative criteria. It is not necessary for implementers to use the specific Schematron encoding in this specification. Implementers **MAY** use any encodings, tools, or languages desired to implement validation schemes for conformance to this specification. However, to conform to the specification, validation schemes **MUST** match the behavior of the reference Schematron implementation. That is, a validator **MUST** find a document valid *if and only if* the reference Schematron implementation would find the document valid according to Whitelist's Schematron rules.

Chapter 2 - Rules

All of the numbered Rules for Whitelist are listed in this section. These rules may depend on patterns defined in the Abstract Patterns section or on variables defined in the Schematron Schema section.

Rules identifiers are all of the format Whitelist-ID-XXXXX, with rule files named Whitelist_ID_XXXXX.sch. Any other heading indicates a supporting file that may influence a rule but is not actually a numbered rule.

2.1 - ../Rules/WLAtomicEnergyMarkings_ID_00001.sch

Rule Description

[WLAtomicEnergyMarkings-ID-00001][Error] Whitelist Validation - @ism:atomicEnergyMarkings combination must be specified in the whitelist. Human Readable: Whitelist Validation Error - ism:atomicEnergyMarkings value must be specified in the whitelist.

Code Description

ism:atomicEnergyMarkings attributes in document must be specified in whitelist. If no values defined in whitelist, then element is not allowed.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLAtomicEnergyMarkings-ID-00001"
             is-a="ValidateTokensExistInWhitelist">
  <sch:param name="context" value="*[@ism:atomicEnergyMarkings]"/>
  <sch:param name="searchTermList" value="@ism:atomicEnergyMarkings"/>
  <sch:param name="list" value="$atomicEnergyMarkingsList_tok"/>
  <sch:param name="errMsg"
             value="' [WLAtomicEnergyMarkings-ID-00001][Error] Whitelist Validation - @ism:atomicEnergyMarkings combination must be specified in the whitelist.'"/>
</sch:pattern>
```

2.2 - ../Rules/WLClassification_ID_00001.sch

Rule Description

[WLClassification-ID-00001][Error] Whitelist Validation - @ism:classification value in document must exist in the whitelist. Human Readable: Whitelist Validation Error - ism:classification attributes in document must be specified in whitelist.

Code Description

ism:Classification attributes in document must be specified in whitelist. If no values defined in whitelist, then element is not allowed.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLClassification-ID-00001" is-a="ValidateTokensExistInWhitelist">
    <sch:param name="context" value="*[@ism:classification]"/>
    <sch:param name="searchTermList" value="@ism:classification"/>
    <sch:param name="list" value="$classificationList_tok"/>
    <sch:param name="errMsg"
        value="' [WLClassification-ID-00001][Error] Whitelist Validation - @ism:classification value in document must exist in the whitelist.'"/>
</sch:pattern>
```

2.3 - ../Rules/WLDisseminationControls_ID_00001.sch

Rule Description

[WLDisseminationControls-ID-00001][Error] Whitelist Validation - each @ism:disseminationControls value in document must exist in the whitelist. Human Readable: Whitelist Validation Error - each @ism:disseminationControls value in document must exist in the whitelist.

Code Description

ism:disseminationControls attributes in document must be specified in whitelist. If no values defined in whitelist, then element is not allowed.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLDisseminationControls-ID-00001"
             is-a="ValidateTokensExistInWhitelist">
  <sch:param name="context" value="*[@ism:disseminationControls]"/>
  <sch:param name="searchTermList" value="@ism:disseminationControls"/>
  <sch:param name="list" value="$disseminationControlsList_tok"/>
  <sch:param name="errMsg"
             value="' [WLDisseminationControls-ID-00001][Error] Whitelist Validation - each @ism:disseminationControls value in document must exist in the whitelist.'"/>
</sch:pattern>
```

2.4 - ../Rules/WLFGIsourceOpen_ID_00001.sch

Rule Description

[WLFGIsourceOpen-ID-00001][Error] Whitelist Validation - each @ism:FGIsourceOpen values in document must exist in whitelist.

Code Description

If FGIsourceOpen/values@type = 'whitelist', then all ism:FGIsourceOpen values must exist in FGIsourceOpen/values. Leaving the values element empty with @type = 'whitelist' means that any value will fail.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLFGIsourceOpen-ID-00001" is-a="ValidateTokensExistInWhitelist">
    <sch:param name="context" value="*[@ism:FGIsourceOpen]"/>
    <sch:param name="searchTermList" value="@ism:FGIsourceOpen"/>
    <sch:param name="list" value="$FGIsourceOpenList_tok"/>
    <sch:param name="errMsg"
        value="" [WLFGIsourceOpen-ID-00001][Error] Whitelist Validation - each @ism:FGIsourceOpen values in document must exist in whitelist.'"/>
</sch:pattern>
```

2.5 - ../Rules/WLFGIsourceProtected_ID_00001.sch

Rule Description

[WLFGIsourceProtected-ID-00001][Error] Whitelist Validation - each @ism:FGIsourceProtected value in document must exist in whitelist.

Code Description

If FGIsourceProtected/values@type = 'whitelist', then all ism:FGIsourceProtected values must exist in FGIsourceProtected/values. Leaving the values element empty with @type = 'whitelist' means that any value will fail.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLFGIsourceProtected-ID-00001"
             is-a="ValidateTokensExistInWhitelist">
  <sch:param name="context" value="*[@ism:FGIsourceProtected]"/>
  <sch:param name="searchTermList" value="@ism:FGIsourceProtected"/>
  <sch:param name="list" value="$FGIsourceProtectedList_tok"/>
  <sch:param name="errMsg"
             value="' [WLFGIsourceProtected-ID-00001][Error] Whitelist Validation - each @ism:FGIsourceProtected value in document must exist in whitelist.'"/>
</sch:pattern>
```

2.6 - ../Rules/WLISMAAttributes_ID_00001.sch

Rule Description

[WLISMAAttributes-ID-00001][Error] Whitelist Validation - each @ism attribute in document must exist in whitelist.

Code Description

If ISMAAttributes/values@type = 'whitelist', then all ism: attribute values must exist in ISMAAttributes/values. Leaving the values element empty with @type = 'whitelist' means that any ism attribute will fail.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLISMAAttribute-ID-00001" is-a="ValidateTokensExistInWhitelist">
    <sch:param name="context" value="*[/*[@*[namespace-uri()='urn:us:gov:ic:ism']]]"/>
    <sch:param name="searchTermList"
        value="util:getSpaceSeparatedStringFromSequence(@*[namespace-uri()='urn:us:gov:ic:ism']/local-name())"/>
    <sch:param name="list" value="$ismAttributesList_tok"/>
    <sch:param name="errMsg"
        value="' [WLISMAAttribute-ID-00001][Error] Whitelist Validation - each @ism: attribute in document must exist in whitelist.'"/>
</sch:pattern>
```

2.7 - ../Rules/WLNtkAccessPolicy_ID_00001.sch

Rule Description

[WLAcessPolicy-ID-00001][Error] NTK AccessPolicy Whitelist Validation - each ntk:AccessPolicy value in document must exist in the whitelist. Human Readable: Whitelist Validation Error: each ntk:AccessPolicy value found in document must be specified in the whitelist.

Code Description

All allowable NTK Profile URIs are defined in whitelist. A list of the allowable values is created from the whitelist configuration xml file; this list is compared against a list each NTK AccessProfile value found in xml document. All document NTK values must exist in the list of valued defined in whitelist.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLNtkAccessPolicy-ID-00001" is-a="AllValuesExistInList">
    <sch:param name="context"
        value="//ntk:RequiresAllOf/ntk:AccessProfileList/ntk:AccessProfile/ntk:AccessPolicy"/>
    <sch:param name="list" value="$policyList"/>
    <sch:param name="errMsg"
        value="' [WLAcessPolicy-ID-00001][Error] NTK AccessPolicy Whitelist Validation - each ntk:AccessPolicy value in document must exist in the whitelist.'" />
</sch:pattern>
```

2.8 - ../Rules/WLNtkAccessPolicy_ID_00002.sch

Rule Description

[WLNtkAccessPolicy-ID-00002][Error] NTK AccessPolicy Whitelist Validation - for ntk:RequiresAnyOf, each ntk:AccessPolicy value in document must exist in the whitelist. Human Readable: Whitelist Validation Error: each ntk:AccessPolicy value found in document must be specified in the whitelist.

Code Description

All allowable NTK Profile URIs are defined in whitelist. A list of the allowable values is created from the whitelist configuration xml file; this list is compared against a list each NTK AccessProfile value found in xml document. All document NTK values must exist in the list of valued defined in whitelist.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLNtkAccessPolicy-ID-00002" is-a="SomeValuesExistInList">
    <sch:param name="context"
               value="//ntk:RequiresAnyOf/ntk:AccessProfileList/ntk:AccessProfile/ntk:AccessPolicy"/>
    <sch:param name="list" value="$policyList"/>
    <sch:param name="errMsg"
               value="' [WLNtkAccessPolicy-ID-00002][Error] NTK AccessPolicy Whitelist Validation - for ntk:RequiresAnyOf, each ntk:AccessPolicy value in document must exist in
the whitelist.'"/>
</sch:pattern>
```

2.9 - ../Rules/WLNtkAccessProfileValue_ID_00001.sch

Rule Description

[WLNtkAccessProfileValue-ID-00001][Error] RequiresAnyOf NTK Whitelist Validation - For each profile within an NTK RequiresAnyOf element, at least one AccessProfileValue must be specified in the whitelist. For urn:us:gov:ic:aces:ntk:restrictive profiles, every AccessProfileValue for that profile must be defined in the whitelist. For urn:us:gov:ic:aces:ntk:permissive profiles, at least one AccessProfileValue must be defined. Human Readable: Whitelist RequiresAnyOf Validation Error: for RequiresAnyOf profiles, at least one combination of AccessPolicy, AccessProfileValue@ntk:vocabulary and ntk:AccessProfileValue from document must be specified for the profile in the whitelist.

Code Description

Utilizes the AnyAccessProfileInWhitelist abstract rule. The abstract rule handles the variations in behavior with the restrictive / permissive profiles, the propin:1 profile with no AccessProfileValues and the ICO profile.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLNtkAccessProfileValue-ID-00001">
    <sch:rule id="WLNtkAccessProfileValue-ID-00001-R1"
        context="//ntk:RequiresAnyOf/ntk:AccessProfileList">
        <sch:assert test="some $docProfile in ./ntk:AccessProfile satisfies ( ( $docProfile/ntk:AccessPolicy = 'urn:us:gov:ic:aces:ntk:ico' ) or ( $docProfile/
ntk:AccessPolicy='urn:us:gov:ic:aces:ntk:propin:1' and not($docProfile[$docProfile/ntk:AccessPolicy='urn:us:gov:ic:aces:ntk:propin:1']/ntk:AccessProfileValue) ) or ( if ($docProfile/
ntk:AccessPolicy != 'urn:us:gov:ic:aces:ntk:permissive') then every $docTerm in for $value in $docProfile[./ntk:AccessPolicy != 'urn:us:gov:ic:aces:ntk:permissive']/ntk:AccessProfileValue
return normalize-space(concat($value/./ntk:AccessPolicy, ' ', $value/@ntk:vocabulary, ' ', $value)) satisfies some $listTerm in $accessProfileValueList satisfies (matches ($docTerm,
$listTerm)) else some $docTerm in for $value in $docProfile[./ntk:AccessPolicy='urn:us:gov:ic:aces:ntk:permissive']/ntk:AccessProfileValue return normalize-space(concat($value/./
ntk:AccessPolicy, ' ', $value/@ntk:vocabulary, ' ', $value)) satisfies some $listTerm in $accessProfileValueListPermissive satisfies (matches ($docTerm, $listTerm)) ) )"
            flag="error"
            role="error">[WLNtkAccessProfileValue-ID-00001][Error] RequiresAnyOf NTK Whitelist Validation - For each profile within an NTK RequiresAnyOf element, at
least one AccessProfileValue must be specified in the whitelist. For urn:us:gov:ic:aces:ntk:restrictive profiles, every AccessProfileValue must be defined in the whitelist. For
urn:us:gov:ic:aces:ntk:permissive profiles, at least one AccessProfileValue must be defined. Document Value(s): [
        <sch:value-of select="util:getCommaSeparatedStringFromSequence(for $value in ./ntk:AccessProfile/ntk:AccessProfileValue return normalize-space(concat($value/./ntk:AccessPolicy, ' ',
$value/@ntk:vocabulary, ' ', $value)))"/>] Whitelist Value(s): [
        <sch:value-of select="util:getCommaSeparatedStringFromSequence($accessProfileValueList)"/>
        <sch:value-of select="util:getCommaSeparatedStringFromSequence($accessProfileValueListPermissive)"/>]
    </sch:assert>
    </sch:rule>
</sch:pattern>
```

2.10 - ./Rules/WLNtkAccessProfileValue_ID_00002.sch

Rule Description

[WLNtkAccessProfileValue-ID-00002][Error] RequiresAllOf NTK Whitelist Validation - For each profile within an NTK RequiresAllOf element, at least one AccessProfileValue must be specified in the whitelist. For urn:us:gov:ic:aces:ntk:restrictive profiles, every AccessProfileValue must be defined in the whitelist. For urn:us:gov:ic:aces:ntk:permissive profiles, at least one AccessProfileValue must be defined. Human Readable: Whitelist RequiresAllOf Validation Error: all combinations of AccessPolicy, AccessProfileValue@ntk:vocabulary and ntk:AccessProfileValue from document must be specified for the profile in the whitelist.

Code Description

Utilizes the AllAccessProfilesInWhitelist abstract rule. The abstract rule handles the variations in behavior with the restrictive / permissive profiles, the propin:1 profile with no AccessProfileValues and the ICO profile.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLNtkAccessProfileValue-ID-00002">
    <sch:rule id="WLNtkAccessProfileValue-ID-00002-R1"
        context="//ntk:RequiresAllOf/ntk:AccessProfileList">
        <sch:assert test="every $docProfile in ./ntk:AccessProfile satisfies ( ( $docProfile/ntk:AccessPolicy = 'urn:us:gov:ic:aces:ntk:ico' ) or ( $docProfile/
ntk:AccessPolicy='urn:us:gov:ic:aces:ntk:propin:1' and not($docProfile[$docProfile/ntk:AccessPolicy='urn:us:gov:ic:aces:ntk:propin:1']/ntk:AccessProfileValue) ) or ( (if ($docProfile/
ntk:AccessPolicy != 'urn:us:gov:ic:aces:ntk:permissive') then every $docTerm in for $value in $docProfile[(./ntk:AccessPolicy != 'urn:us:gov:ic:aces:ntk:permissive')]/ntk:AccessProfileValue
return normalize-space(concat($value/./ntk:AccessPolicy, ' ', $value/@ntk:vocabulary, ' ', $value)) satisfies some $listTerm in $accessProfileValueList satisfies (matches ($docTerm,
$listTerm)) else some $docTerm in for $value in $docProfile[(./ntk:AccessPolicy = 'urn:us:gov:ic:aces:ntk:permissive')]/ntk:AccessProfileValue return normalize-space(concat($value/./
ntk:AccessPolicy, ' ', $value/@ntk:vocabulary, ' ', $value)) satisfies some $listTerm in $accessProfileValueListPermissive satisfies (matches ($docTerm, $listTerm)) ) ) )"
            flag="error"
            role="error">[WLNtkAccessProfileValue-ID-00002][Error] RequiresAllOf NTK Whitelist Validation - For each profile within an NTK RequiresAllOf element, at
least one AccessProfileValue must be specified in the whitelist. For urn:us:gov:ic:aces:ntk:restrictive profiles, every AccessProfileValue must be defined in the whitelist. For
urn:us:gov:ic:aces:ntk:permissive profiles, at least one AccessProfileValue must be defined. Document Value(s): [
        <sch:value-of select="util:getCommaSeparatedStringFromSequence(for $value in ./ntk:AccessProfile/ntk:AccessProfileValue return normalize-space(concat($value/@ntk:vocabulary, ' ',
$value)))"/>] Whitelist Value(s): [
        <sch:value-of select="util:getCommaSeparatedStringFromSequence($accessProfileValueList)"/>
        <sch:value-of select="util:getCommaSeparatedStringFromSequence($accessProfileValueListPermissive)"/>]
    </sch:assert>
    </sch:rule>
</sch:pattern>
```

2.11 - ../Rules/WLNtkAccessProfileValue_ID_00003.sch

Rule Description

[WLNtkAccessProfileValue-ID-00003][Error] NTK Minimum Values Validation - at a minimum, each whitelist profile value, defined by combination of AccessProfileValue@ntk:vocabulary and ntk:AccessProfileValue, must be found in document. Human Readable: At a minimum, each whitelist profile value, defined by combination of AccessProfileValue@ntk:vocabulary and ntk:AccessProfileValue, must be found in document. Additional profile values can exist.

Code Description

Minimum set of NTK Profile values that must exist in document being evaluated. Minimum value entry is a combination of required profile values as specified by a combination of AccessProfileValue@ntk:vocabulary and ntk:AccessProfileValue. A list of required values is created from the configuration xml file for the profile. Each NTK profile value found in xml document must exist in this list along with any other profiles.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLNtkAccessProfileValue-ID-00003">
    <sch:rule id="WLNtkAccessProfileValue-ID-00003-R1"
        context="//ntk:AccessProfileList/ntk:AccessProfile[some $policyTerm in ../ntk:AccessPolicy satisfies some $configPolicyTerm in $profilesWithMinValues satisfies
matches($policyTerm, $configPolicyTerm) ]">
        <sch:assert test="(some $listTerm in $accessProfileMinValueList satisfies some $docTerm in for $value in .[(. != 'urn:us:gov:ic:aces:ntk:permissive')]/
ntk:AccessProfileValue return normalize-space(concat($value/../../ntk:AccessPolicy, ' ', $value/@ntk:vocabulary, ' ', $value)) satisfies (matches ($docTerm, $listTerm)) )"
            flag="error"
            role="error">[WLNtkAccessProfileValue-ID-00003][Error] NTK Minimum Values Validation - at a minimum, each whitelist profile value, defined by combination of
AccessProfileValue@ntk:vocabulary and ntk:AccessProfileValue, must be found in document. Document NTK Profiles[
        <sch:value-of select="util:getCommaSeparatedStringFromSequence( ../ntk:AccessPolicy )"/>] Profiles with Minimum Values Defined: [
        <sch:value-of select="util:getCommaSeparatedStringFromSequence($profilesWithMinValues)"/>] Document Value(s): [
        <sch:value-of select="util:getCommaSeparatedStringFromSequence(for $value in ../ntk:AccessProfileValue return normalize-space(concat($value/../../ntk:AccessPolicy, ' ', $value/
@ntk:vocabulary, ' ', $value)))"/>] Minimum required value(s): [
        <sch:value-of select="util:getCommaSeparatedStringFromSequence($accessProfileMinValueList)"/>]
    </sch:assert>
    </sch:rule>
</sch:pattern>
```

2.12 - ./Rules/WLNtkVocabulary_ID_00001.sch

Rule Description

[WLNtkVocabulary-ID-00001][Error] NTK Vocabulary Type Whitelist Validation - each ntk:AccessProfileValue@vocabulary value in document must exist in the whitelist.. Human Readable: Whitelist Validation Error: each ntk:AccessPolicy value found in document must be specified in the whitelist.

Code Description

All allowable NTK Vocabulary URIs are defined in whitelist. A list of the allowable values is created from the whitelist configuration xml file; this list is compared against a list each NTK AccessProfileValue@vocabulary value found in xml document. All document NTK values must exist in the list of valued defined in whitelist.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLNtkVocabulary-ID-00001" is-a="AllValuesExistInList">
    <sch:param name="context"
               value="//ntk:RequiresAllOf/ntk:AccessProfileList/ntk:AccessProfile/ntk:AccessProfileValue/@ntk:vocabulary"/>
    <sch:param name="list" value="$vocabList"/>
    <sch:param name="errMsg"
               value="' [WLNtkVocabulary-ID-00001][Error] NTK Vocabulary Type Whitelist Validation - each ntk:AccessProfileValue@vocabulary value in document must exist in the
whitelist.'"/>
</sch:pattern>
```

2.13 - .//Rules/WLNtkVocabulary_ID_00002.sch

Rule Description

[WLNtkVocabulary-ID-00002][Error] NTK Vocabulary Type Whitelist Validation - for ntk:RequiresAnyOf, each ntk:AccessProfileValue@vocabulary value in document must exist in the whitelist. Human Readable: Whitelist Validation Error: each ntk:AccessPolicy value found in document must be specified in the whitelist.

Code Description

All allowable NTK Vocabulary URIs are defined in whitelist. A list of the allowable values is created from the whitelist configuration xml file; this list is compared against a list each NTK AccessProfileValue@vocabulary value found in xml document. All document NTK values must exist in the list of valued defined in whitelist. Pass one list to SomeValuesExistInList abstract rule - list of all NTK Vocabulary values defined in config file

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLNtkVocabulary-ID-00002" is-a="SomeValuesExistInList">
    <sch:param name="context"
        value="//ntk:RequiresAnyOf/ntk:AccessProfileList/ntk:AccessProfile/ntk:AccessProfileValue/@ntk:vocabulary"/>
    <sch:param name="list" value="$vocabList"/>
    <sch:param name="errMsg"
        value="' [WLNtkVocabulary-ID-00002][Error] NTK Vocabulary Type Whitelist Validation - for ntk:RequiresAnyOf, each ntk:AccessProfileValue@vocabulary value in
document must exist in the whitelist.'" />
</sch:pattern>
```

2.14 - ./Rules/WLReleaseableTo_ID_00001.sch

Rule Description

[WLReleaseableTo-ID-00001][Error] Whitelist Validation - each @ism:releasableTo value in document must exist in whitelist.

Code Description

If ReleaseableTo/values@type = 'whitelist', then all ism:releasableTo values must exist in ReleaseableTo/values. Tetragraphs will be decomposed. Leaving the values element empty with @type = 'whitelist' means that any value will fail.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLReleaseableTo-ID-00001"
    is-a="ValidateDecomposableTokensExistInWhitelist">
    <sch:param name="context" value="*[@ism:releasableTo]"/>
    <sch:param name="searchTermList" value="@ism:releasableTo"/>
    <sch:param name="list" value="$releasableToList_tok"/>
    <sch:param name="errMsg"
        value="' [WLReleaseableTo-ID-00001][Error] Whitelist Validation - each @ism:releasableTo value in document must exist in whitelist.'"/>
</sch:pattern>
```

2.15 - ./Rules/WLReleaseableTo_ID_00002.sch

Rule Description

[WLReleaseableTo-ID-00002][Error] Whitelist Validation - if minimumValues is defined for a minimum set of relTo values, then ensure that @ism:releaseableTo values include each member of minimum values.

Code Description

If config file ReleaseableTo element includes the minimumValues child, then each instance of ism:releaseableTo in the document must include each member listed in minimumValues.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLReleaseableTo-ID-00002">
    <sch:rule id="WLReleaseableTo-ID-00002-R1" context="*[@ism:releaseableTo]">
        <sch:let name="searchTermList" value="@ism:releaseableTo"/>
        <sch:assert test="every $Term in for $term in $releaseableToMinList_tok return util:expandDecomposableTetras($term) satisfies some $searchTerm in for $term in
$searchTermList return util:expandDecomposableTetras($term) satisfies (matches (normalize-space($Term), concat('^', $searchTerm, '$')))"
            flag="error"
            role="error">[WLReleaseableTo-ID-00002][Error] Whitelist Validation - if minimumValues is defined for a minimum set of relTo values, then ensure that
@ism:releaseableTo values include each member of minimum values. Failed whitelist check. Document releaseableTo value(s): [
        <sch:value-of select="util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras($searchTermList))"/>] Minimum required releaseableTo value(s): [
        <sch:value-of select="util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras( $releaseableToMinList_tok ))"/>]
    </sch:assert>
    </sch:rule>
</sch:pattern>
```

2.16 - ../Rules/WLSARIdentifier_ID_00001.sch

Rule Description

[WLSARIdentifier-ID-00001][Error] Whitelist Validation - each @ism:SARIdentifier attribute value in document must exist in the whitelist. Human Readable: Whitelist Validation Error - each @ism:SARIdentifier attribute value in document must exist in the whitelist.

Code Description

ism:SARIdentifier attributes in document must be specified in whitelist. If no values defined in whitelist, then element is not allowed.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLSARIdentifier-ID-00001" is-a="ValidateTokensExistInWhitelist">
    <sch:param name="context" value="*[@ism:SARIdentifier]"/>
    <sch:param name="searchTermList" value="@ism:SARIdentifier"/>
    <sch:param name="list" value="$SARIdentifierList_tok"/>
    <sch:param name="errMsg"
        value="' [WLSARIdentifier-ID-00001] Whitelist Validation - each @ism:SARIdentifier attribute value in document must exist in the whitelist.'" />
</sch:pattern>
```

2.17 - ./Rules/WLSCIcontrols_ID_00001.sch

Rule Description

[WLSCIcontrols-ID-00001][Error] Whitelist Validation - each @ism:SCIcontrols each combination must be specified in the whitelist. Human Readable: SCIcontrols Validation Error: each @ism:SCIcontrols value in document must exist in whitelist.

Code Description

ism:SCIcontrols attributes in document must be specified in whitelist. If no values defined in whitelist, then element is not allowed.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLSCIcontrols-ID-00001" is-a="ValidateTokensExistInWhitelist">
  <sch:param name="context" value="*[@ism:SCIcontrols]"/>
  <sch:param name="searchTermList" value="@ism:SCIcontrols"/>
  <sch:param name="list" value="$SCIControlsList_tok"/>
  <sch:param name="errMsg"
    value="" [WLSCIcontrols-ID-00001][Error] Whitelist Validation - each @ism:SCIcontrols each combination must be specified in the whitelist.'"/>
</sch:pattern>
```

2.18 - ./Rules/WLdisplayOnlyTo_ID_00001.sch

Rule Description

[WLdisplayOnlyTo-ID-00001][Error] Whitelist Validation - each @ism:displayOnlyTo value in document must exist in whitelist.

Code Description

If DisplayOnlyTo/values@type = 'whitelist', then all ism:displayOnlyTo values must exist in DisplayOnlyTo/values. Tetragraphs will be decomposed. Leaving the values element empty with @type = 'whitelist' means that any value will fail.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLdisplayOnlyTo-ID-00001"
    is-a="ValidateDecomposableTokensExistInWhitelist">
    <sch:param name="context" value="*[@ism:displayOnlyTo]"/>
    <sch:param name="searchTermList" value="@ism:displayOnlyTo"/>
    <sch:param name="list" value="$displayOnlyToList_tok"/>
    <sch:param name="errMsg"
        value="' [WLdisplayOnlyTo-ID-00001][Error] Whitelist Validation - each @ism:displayOnlyTo value in document must exist in whitelist.'"/>
    </sch:pattern>
```

2.19 - .//Rules/WLnonICmarkings_ID_00001.sch

Rule Description

[WLnonICmarkings-ID-00001][Error] Whitelist validation: each @ism:nonICmarkings value in document must exist in whitelist. Human Readable: Whitelist Validation Error: each @ism:nonICmarkings value in document must exist in whitelist.

Code Description

ism:SClcontrols attributes in document must be specified in whitelist. If no values defined in whitelist, then element is not allowed.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLnonICmarkings-ID-00001" is-a="ValidateTokensExistInWhitelist">
    <sch:param name="context" value="*[@ism:nonICmarkings]"/>
    <sch:param name="searchTermList" value="@ism:nonICmarkings"/>
    <sch:param name="list" value="$nonICmarkingsList_tok"/>
    <sch:param name="errMsg"
        value="' [WLnonICmarkings-ID-00001][Error] Whitelist validation: each @ism:nonICmarkings value in document must exist in whitelist.'"/>
</sch:pattern>
```

2.20 - ../Rules/WLnonUSControls_ID_00001.sch

Rule Description

[WLnonUSControls-ID-00001][Error] Whitelist Validation - each @ism:nonUSControls attribute value in document must exist in the whitelist. Human Readable: Whitelist Validation Error - each @ism:nonUSControls attribute value in document must exist in the whitelist.

Code Description

ism:nonUSControls attributes in document must be specified in whitelist. If no values defined in whitelist, then element is not allowed.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLnonUSControls-ID-00001" is-a="ValidateTokensExistInWhitelist">
    <sch:param name="context" value="*[@ism:nonUSControls]"/>
    <sch:param name="searchTermList" value="@ism:nonUSControls"/>
    <sch:param name="list" value="$nonUSControlsList_tok"/>
    <sch:param name="errMsg"
        value="' [WLnonUSControls-ID-00001] Whitelist Validation - each @ism:nonUSControls attribute value in document must exist in the whitelist.'" />
</sch:pattern>
```

2.21 - ../Rules/WLownerProducer_ID_00001.sch

Rule Description

[WLownerProducer-ID-00001][Error] Whitelist Validation - each @ism:ownerProducer value in document must exist in whitelist.

Code Description

If OwnerProducer/values@type = 'whitelist', then all ism:ownerProducer values must exist in OwnerProducer/values. Leaving the values element empty with @type = 'whitelist' means that any value will fail.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLownerProducer-ID-00001" is-a="ValidateTokensExistInWhitelist">
    <sch:param name="context"
        value="*[( $ownerProducerListType = 'whitelist') and @ism:ownerProducer]"/>
    <sch:param name="searchTermList" value="@ism:ownerProducer"/>
    <sch:param name="list" value="$ownerProducerList_tok"/>
    <sch:param name="errMsg"
        value="' [WLownerProducer-ID-00001][Error] Whitelist Validation - each @ism:ownerProducer value in document must exist in whitelist.'"/>
</sch:pattern>
```

2.22 - ../Rules/WLownerProducer_ID_00002.sch

Rule Description

[WLownerProducer-ID-00002][Error] Blacklist Validation - @ism:ownerProducer values in document must not exist in blacklist.

Code Description

If OwnerProducer/values@type = 'blacklist', then none of the ism:ownerProducer values must exist in OwnerProducer/values Leaving the values element empty with @type = 'blacklist' means that any value will pass.

Schematron Code

```
<?ICEA pattern?>
<sch:pattern id="WLownerProducer-ID-00002" is-a="ValidateTokenValuesNotInBlacklist">
    <sch:param name="context"
        value="*[( $ownerProducerListType = 'blacklist') and @ism:ownerProducer]"/>
    <sch:param name="searchTermList" value="@ism:ownerProducer"/>
    <sch:param name="list" value="$ownerProducerList_tok"/>
    <sch:param name="errMsg"
        value="' [WLownerProducer-ID-00002][Error] Blacklist Validation - @sm:ownerProducer values in document must not exist in blacklist.'"/>
</sch:pattern>
```

Chapter 3 - Abstract Patterns

All of the Abstract Patterns for Whitelist are listed in this section. These patterns may depend on variables defined in the Schematron Schema section.

3.1 - ./Lib/AllValuesExistInList.sch

Code Description

This abstract pattern verifies that a value at a given context matches fome value in a list using regular expression matching. The calling rule must pass the context, search term list, attribute value to check, and an error message.

Schematron Code

```
<!-- Notices - Distribution Notice:
      This document has been approved for Public Release and is available for use without restriction.
-->
<!--
  This abstract pattern verifies that all values of a given context match a value in a given list. The
  calling rule must pass the context that will provide a list of values to check, list of values from configuration
  file to match, and an error message.

  $context      := the context in which the search values exists
  $list          := the list in which the target element value should exist
  $errMsg        := the error message text to display when the assertion fails

  Example usage:
  Inside an ntk:RequiresAllOf structure, ensure that all ntk:AccessProfile values exist in whitelist

  <sch:pattern is-a="AllValuesExistsInList" id="WLAccessPolicy-ID-00001" xmlns:sch="http://purl.oclc.org/dsdl/schematron">
    <sch:param name="context"
      value="//ntk:RequiresAnyOf/ntk:AccessProfileList/ntk:AccessProfile/ntk:AccessPolicy"/>
    <sch:param name="list" value="$policyList"/>
    <sch:param name="errMsg" value="'
      [WLAccessPolicy-ID-00001][Error] NTK AccessPolicy Whitelist Validation -
      all ntk:AccessPolicy values in document must exist in the whitelist.'" />
  </sch:pattern>

-->

<sch:pattern abstract="true" id="AllValuesExistInList">
  <sch:rule id="AllValuesExistInList-R1" context="$context"><!--<sch:let name="fullSearchTermList" value="$context"/>-->
<!--<sch:let name="initialItem" value="util:getFirstItemFromSequence($list)"/>-->

  <sch:assert test="( (matches(util:getFirstItemFromSequence($list),'\\*')) or ( every $searchTerm in $context satisfies some $term in $list satisfies (matches ($searchTerm , $term)) ) )"
    flag="error"
    role="error">
    <sch:value-of select="$errMsg"/>Document Value(s): [
  <sch:value-of select="util:getCommaSeparatedStringFromSequence($context)"/>] Whitelist Value(s): [
  <sch:value-of select="util:getCommaSeparatedStringFromSequence($list)"/>]
</sch:assert>

  </sch:rule>
</sch:pattern>
```

3.2 - ../Lib/SomeValuesExistInList.sch

Code Description

This abstract pattern verifies that the values of child elements at a given context match some value in a list. The calling rule must pass the context, child element name to build search term list, list of values from configuration file, and an error message.

Schematron Code

```
<!-- Notices - Distribution Notice:
      This document has been approved for Public Release and is available for use without restriction.
-->
<!--
  This abstract pattern verifies that some values of a given context match a value in a given list. The
  calling rule must pass the context that will provide a list of values to check, list of values from configuration
  file to match, and an error message.

  $context      := the context in which the search values exists
  $list          := the list in which the target element value should exist
  $errMsg        := the error message text to display when the assertion fails

  Example usage:
  Inside a ntk:RequiresAnyOf structure, ensure that at least one ntk:AccessProfile value exists in whitelist.

  <sch:pattern is-a="ValueExistsInList" id="IWLAccessPolicy-ID-00002" xmlns:sch="http://purl.oclc.org/dsdl/schematron">
    <sch:param name="context" value="//ntk:RequiresAnyOf/ntk:AccessProfileList/ntk:AccessProfile/ntk:AccessPolicy"/>
    <sch:param name="list" value="$policyList"/>
    <sch:param name="errMsg" value="'
      [WLAccessPolicy-ID-00002][Error] NTK AccessPolicy Whitelist Validation -
      at least one ntk:AccessPolicy value in document must exist in the whitelist.'
    '"/>
  </sch:pattern>

-->

<sch:pattern abstract="true" id="SomeValuesExistInList">
  <sch:rule id="SomeValuesExistInList-R1" context="$context">
    <sch:assert test="( (matches(util:getFirstItemFromSequence($list),'\\*')) or (some $searchTerm in $context satisfies some $term in $list satisfies (matches
($searchTerm , $term))) )"
      flag="error"
      role="error">
      <sch:value-of select="$errMsg"/>Document Value(s): [
    <sch:value-of select="util:getCommaSeparatedStringFromSequence($context)"/>] Whitelist Value(s): [
    <sch:value-of select="util:getCommaSeparatedStringFromSequence($list)"/>]
  </sch:assert>
  </sch:rule>
</sch:pattern>
```

3.3 - ./Lib/ValidateDecomposableTokensExistInWhitelist.sch

Code Description

This abstract pattern checks to see if an attribute of an element exists in a list or matches the pattern defined by the list. The calling rule must pass the context, search term list, attribute value to check, and an error message.

Schematron Code

```
<!--
  Based on XmlEncodings/Schematron/ISM/Lib/ValidateTokenValuesExistenceInList.sch.  Changed Failure message to be display tokens
  being checked and whitelist tokens in error message.

  This abstract pattern checks to see if an attribute of an element exists in a list or matches the pattern defined by the list.

  $context      := the context in which the searchValue exists
  $searchTermList := the set of values which you want to verify is in the list
  $list          := the list in which to search for the searchValue
  $errMsg        := the error message text to display when the assertion fails

  Example usage:
  <sch:pattern is-a="ValidateDecomposableTokensExistInWhitelist" id="WLReleaseableTo_ID_00001" xmlns:sch="http://purl.oclc.org/dsdl/schematron">
    <sch:param name="context"      value="*[@ism:releasableTo]"/>
    <sch:param name="searchTermList" value="@ism:releasableTo"/>
    <sch:param name="list" value="$releasableToList_tok"/>
    <sch:param name="errMsg"
      value="' [WLReleaseableTo-ID-00001][Error] Whitelist Validation -
      each @ism:releasableTo value in document must exist in whitelist.'"/>
  </sch:pattern>

-->

<sch:pattern abstract="true" id="ValidateDecomposableTokensExistInWhitelist">
  <sch:rule id="ValidateDecomposableTokensExistInWhitelist-R1" context="$context">
    <sch:assert test="( ( matches(util:getFirstItemFromSequence($list),'\*') ) or (every $searchTerm in
tokenize(util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras($searchTermList)), ' ') satisfies some $Term in
tokenize(util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras($list)), ' ') satisfies (matches (normalize-space($searchTerm), normalize-space($Term))) ) )"
      flag="error"
      role="error">
      <sch:value-of select="$errMsg"/>Document Value(s): [
    <sch:value-of select="util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras($searchTermList))"/>] Whitelist Value(s): [
    <sch:value-of select="util:getCommaSeparatedStringFromSequence($list)"/>]
  </sch:assert>
  </sch:rule>
</sch:pattern>
```

3.4 - ./Lib/ValidateTokenValuesNotInBlacklist.sch

Code Description

This abstract pattern checks to see if an attribute of an element exists in a list or matches the pattern defined by the list. The calling rule must pass the context, search term list, attribute value to check, and an error message.

Schematron Code

```
<!--
  This abstract pattern checks to see if an attribute of an element exists in a blacklist or matches the pattern defined by the list.

  $context      := the context in which the searchValue exists
  $searchTermList := the set of values which verify against the blacklist
  $list          := the blacklist values
  $errMsg        := the error message text to display when the assertion fails

  <sch:param name="context"      value="*[( $FGISourceProtectedListType = 'blacklist' ) and @ism:FGISourceProtected]"/>
  <sch:param name="searchTermList" value="@ism:FGISourceProtected"/>
  <sch:param name="list" value="$FGISourceProtectedList_tok"/>
  <sch:param name="errMsg"
    value=""      [WLFGISourceProtected-ID-00002][Error] Blacklist Validation -  @ism:FGISourceProtected values in document must not exist in blacklist.'"/>
-->

<sch:pattern abstract="true" id="ValidateTokenValuesNotInBlacklist">
  <sch:rule id="ValidateTokenValuesNotInBlacklist-R1" context="$context">
    <sch:assert test="not(util:containsAnyOfTheTokens( (normalize-space(string(util:getSpaceSeparatedStringFromSequence($searchTermList)))), ($list)))"
      flag="error"
      role="error">
      <sch:value-of select="$errMsg"/>Document Value(s): [
    <sch:value-of select="util:getSpaceSeparatedStringFromSequence($searchTermList)"/>] Blacklist Value(s): [
    <sch:value-of select="util:getSpaceSeparatedStringFromSequence($list)"/>]
  </sch:assert>
  </sch:rule>
</sch:pattern>
```

3.5 - ./Lib/ValidateTokensExistInWhitelist.sch

Code Description

This abstract pattern checks to see if an attribute of an element exists in a list or matches the pattern defined by the list. The calling rule must pass the context, search term list, attribute value to check, and an error message.

Schematron Code

```
<!--

  This abstract pattern checks to see if an all attributes of an list exist in a whitelist.

  $context      := the context in which the searchValue exists
  $searchTermList := the set of values which you want to verify is in the list
  $list          := the list in which to search for the searchValue
  $errMsg        := the error message text to display when the assertion fails

  Example usage:
  <sch:pattern is-a="ValidateTokensExistInWhitelist" id="WLReleaseableTo_ID_00001" xmlns:sch="http://purl.oclc.org/dsdl/schematron">
    <sch:param name="context" value="*[@ism:releasableTo]"/>
    <sch:param name="searchTermList" value="@ism:releasableTo"/>
    <sch:param name="list" value="$releasableToList_tok"/>
    <sch:param name="errMsg"
      value="' [WLReleaseableTo-ID-00001][Error] Whitelist Validation - each @ism:releasableTo value in document must exist in whitelist.'"/>
  </sch:pattern>

-->

<sch:pattern abstract="true" id="ValidateTokensExistInWhitelist">
  <sch:rule id="ValidateTokensExistInWhitelist-R1" context="$context">
    <sch:assert test="(matches(util:getFirstItemFromSequence($list),'\*')) or ( ( matches(util:getFirstItemFromSequence($list),'\*') ) or (every $searchTerm in
tokenize(util:getSpaceSeparatedStringFromSequence($searchTermList), ' ') satisfies some $Term in $list satisfies (matches (normalize-space($searchTerm), concat('^', $Term ,'$')) ) )"
      flag="error"
      role="error">
      <sch:value-of select="$errMsg"/>Document Value(s): [
    <sch:value-of select="util:getCommaSeparatedStringFromSequence($searchTermList)"/>] Whitelist Value(s): [
    <sch:value-of select="util:getCommaSeparatedStringFromSequence($list)"/>]
  </sch:assert>
  </sch:rule>
</sch:pattern>
```

Chapter 4 - Schematron Schema

The top level Schematron file for Whitelist is in this section. This file imports all of the others and also defines many global variables they are all dependent on.

4.1 - ./Whitelist_XML.sch

Code Description

This is the root file for the Whitelist Schematron rule set. It loads all of the required CVEs, declares some variables and includes all of the Rule .sch files.

Schematron Code

```
<!--UNCLASSIFIED-->
<!--      Notices - Distribution Notice:
      This document has been approved for Public Release and is available for use without restriction.
-->

<sch:schema queryBinding="xslt2">
    <sch:ns uri="urn:us:gov:ic:ism" prefix="ism"/>
    <sch:ns uri="urn:us:gov:ic:ntk" prefix="ntk"/>
    <sch:ns uri="urn:us:gov:ic:taxonomy:catt:tetragraph" prefix="catt"/>
    <sch:ns uri="urn:us:gov:ic:cve" prefix="cve"/>
    <sch:ns uri="deprecated:value:function" prefix="dvf"/>
    <sch:ns uri="urn:us:gov:ic:ism:xsl:util" prefix="util"/>
    <!-- (U) Abstract Patterns -->

<sch:include href="./Lib/ValidateDecomposableTokensExistInWhitelist.sch"/>
<sch:include href="./Lib/ValidateTokensExistInWhitelist.sch"/>
<sch:include href="./Lib/ValidateTokenValuesNotInBlacklist.sch"/>
<sch:include href="./Lib/AllValuesExistInList.sch"/>
<sch:include href="./Lib/SomeValuesExistInList.sch"/>
<!-- (U) Whitelist Configuration File -->

<sch:let name="configXML" value="document('./whitelist_config.xml')"/>
<!-- (U) Resources -->

<sch:let name="ismAttributesList"
    value="$configXML//Whitelist/ISMAAttributes/values"/>
<sch:let name="ismAttributesListType"
    value="$configXML//Whitelist/ISMAAttributes/values/@type"/>
<sch:let name="classificationList"
    value="$configXML//Whitelist/Classification/values"/>
<sch:let name="disseminationControlsList"
    value="$configXML//Whitelist/DisseminationControls/values"/>
<sch:let name="releasableToList"
    value="util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras($configXML//Whitelist/ReleasableTo/values))"/>
<sch:let name="releasableToMinList"
    value="util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras($configXML//Whitelist/ReleasableTo/minimumValues))"/>
<sch:let name="releasableToListType"
    value="$configXML//Whitelist/ReleasableTo/values/@type"/>
<sch:let name="ownerProducerList"
    value="$configXML//Whitelist/OwnerProducer/values"/>
<sch:let name="ownerProducerListType"
    value="$configXML//Whitelist/OwnerProducer/values/@type"/>
<sch:let name="atomicEnergyMarkingsList"
    value="$configXML//Whitelist/AtomicEnergyMarkings/values"/>
<sch:let name="nonICmarkingsList"
    value="$configXML//Whitelist/NonICmarkings/values"/>
<sch:let name="SCIControlsList" value="$configXML//Whitelist/SCIControls/values"/>
<sch:let name="displayOnlyToList"
    value="util:getSpaceSeparatedStringFromSequence(util:expandDecomposableTetras($configXML//Whitelist/DisplayOnlyTo/values))"/>
<sch:let name="displayOnlyToListType"
    value="$configXML//Whitelist/DisplayOnlyTo/values/@type"/>
```

```

<sch:let name="FGISourceOpenList"
  value="$configXML//Whitelist/FGISourceOpen/values"/>
<sch:let name="FGISourceOpenListType"
  value="$configXML//Whitelist/FGISourceOpen/values/@type"/>
<sch:let name="FGISourceProtectedList"
  value="$configXML//Whitelist/FGISourceProtected/values"/>
<sch:let name="FGISourceProtectedListType"
  value="$configXML//Whitelist/FGISourceProtected/values/@type"/>
<sch:let name="nonUSControlsList"
  value="$configXML//Whitelist/NonUSControls/values"/>
<sch:let name="SARIdentifierList"
  value="$configXML//Whitelist/SARIdentifier/values"/>
<sch:let name="policyList" value="$configXML//Whitelist/NtkPolicy/value"/>
<sch:let name="vocabList" value="$configXML//Whitelist/NtkVocabulary/value"/>
<sch:let name="accessProfileValueList"
  value="for $value in $configXML//Whitelist/NtkAccessProfile[@accessPolicy != 'urn:us:gov:ic:aces:ntk:permissive']/profile return normalize-space(concat($value/../../@accessPolicy, ' ', $value/@vocabulary, ' ', $value/@profileValue))"/>
  <sch:let name="accessProfileValueListPermissive"
    value="for $value in $configXML//Whitelist/NtkAccessProfile[@accessPolicy = 'urn:us:gov:ic:aces:ntk:permissive']/profile return normalize-space(concat($value/../../@accessPolicy, ' ', $value/@vocabulary, ' ', $value/@profileValue))"/>
  <sch:let name="profilesWithMinValues"
    value="$configXML//Whitelist/NtkAccessProfile[count(./minimumValues/profile/@profileValue) > 0]/@accessPolicy"/>
  <sch:let name="accessProfileMinValueList"
    value="for $value in $configXML//Whitelist/NtkAccessProfile[@accessPolicy != 'urn:us:gov:ic:aces:ntk:restrictive']/minimumValues/profile return normalize-space(concat($value/../../@accessPolicy, ' ', $value/@vocabulary, ' ', $value/@profileValue))"/>
  <sch:let name="ismAttributesList_tok"
    value="tokenize(normalize-space(string($ismAttributesList)), ' ')">
  <sch:let name="classificationList_tok"
    value="tokenize(normalize-space(string($classificationList)), ' ')">
  <sch:let name="disseminationControlsList_tok"
    value="tokenize(normalize-space(string($disseminationControlsList)), ' ')">
  <sch:let name="releasableToList_tok"
    value="for $value in $releasableToList return normalize-space(concat($value, ' '))"/>
  <sch:let name="releasableToMinList_tok"
    value="for $value in $releasableToMinList return normalize-space(concat($value, ' '))"/>
  <sch:let name="ownerProducerList_tok"
    value="tokenize(normalize-space(string($ownerProducerList)), ' ')">
  <sch:let name="atomicEnergyMarkingsList_tok"
    value="tokenize(normalize-space(string($atomicEnergyMarkingsList)), ' ')">
  <sch:let name="SCIControlsList_tok"
    value="tokenize(normalize-space(string($SCIControlsList)), ' ')">
  <sch:let name="nonICmarkingsList_tok"
    value="tokenize(normalize-space(string($nonICmarkingsList)), ' ')">
  <sch:let name="displayOnlyToList_tok"
    value="for $value in $displayOnlyToList return normalize-space(concat($value, ' '))"/>
  <sch:let name="FGISourceOpenList_tok"
    value="tokenize(normalize-space(string($FGISourceOpenList)), ' ')">
  <sch:let name="FGISourceProtectedList_tok"
    value="tokenize(normalize-space(string($FGISourceProtectedList)), ' ')">
  <sch:let name="nonUSControlsList_tok"
    value="tokenize(normalize-space(string($nonUSControlsList)), ' ')">
  <sch:let name="SARIdentifierList_tok"
    value="tokenize(normalize-space(string($SARIdentifierList)), ' ')">
<!-- ISMCAT Dependencies for decomposition -->

```

```

<sch:let name="catt"
    value="document(' ../../Taxonomy/ISMCAT/TetragraphTaxonomy.xml ' )"/>
<sch:let name="cattMappings" value="$catt//catt:Tetragraph"/>
<sch:let name="decomposableTetraElems"
    value="$cattMappings[@decomposable[. = 'Yes']]/>
<sch:let name="decomposableTetras"
    value="$decomposableTetraElems/catt:TetraToken/text()"/>
<!--*****-->
<!-- (U) Custom XSLT functions -->
<!--*****-->
<!-- Returns true if all token in the attribute value match at least one token in the provided list.-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="util:containsOnlyTheTokens"
    as="xs:boolean">
    <xsl:param name="attribute"/>
    <xsl:param name="tokenList" as="xs:string*" />
    <xsl:value-of select="every $attrToken in tokenize(normalize-space(string($attribute)), ' ') satisfies $attrToken = $tokenList"/>
</xsl:function>
<!-- Return first item in a list of values -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="util:getFirstItemFromSequence"
    as="xs:string">
    <xsl:param name="listValues"/>
    <xsl:variable name="StringValue">
        <xsl:for-each select="$listValues">
            <xsl:value-of select="current()"/>
            <xsl:if test="position() = 1">
                <xsl:value-of select="current()"/>
            </xsl:if>
        </xsl:for-each>
    </xsl:variable>
    <xsl:value-of select="normalize-space(string($StringValue))"/>
</xsl:function>
<!--
    Return a list of values as a space delimited string from a sequence of tokens
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="util:getSpaceSeparatedStringFromSequence"
    as="xs:string">
    <xsl:param name="attrValues"/>
    <xsl:variable name="StringValues">
        <xsl:for-each select="$attrValues">
            <xsl:value-of select="current()"/>
            <xsl:if test="position() != last()">
                <xsl:value-of select="' '"/>
            </xsl:if>
        </xsl:for-each>
    </xsl:variable>
    <xsl:value-of select="normalize-space(string($StringValues))"/>
</xsl:function>

```

```
<!--
  Return a list of values as a comma delimited string from a sequence of tokens
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:getCommaSeparatedStringFromSequence"
              as="xs:string">
  <xsl:param name="attrValues"/>
  <xsl:variable name="StringValues">
    <xsl:for-each select="$attrValues">
      <xsl:value-of select="current()"/>
      <xsl:if test="position() != last()">
        <xsl:value-of select="', '"/>
      </xsl:if>
    </xsl:for-each>
  </xsl:variable>
  <xsl:value-of select="normalize-space(string($StringValues))"/>
</xsl:function>
<!-- Returns the sequence of country codes that correspond to the given $tetra -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:getCountriesForTetra"
              as="xs:string*">
  <xsl:param name="tetra" as="xs:string"/>
  <xsl:sequence select="$decomposableTetraElems[catt:TetraToken/text() = $tetra]/catt:Membership/catt:Country/text()"/>
</xsl:function>
<!--
  Returns true if the attribute @ism:excludeFromRollup is present and evaluates to 'true'
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:contributesToRollup"
              as="xs:boolean">
  <xsl:param name="context"/>
  <xsl:value-of select="not(string($context/@ism:excludeFromRollup) = 'true')"/>
</xsl:function>
<!-- Returns normalized $value with a preceding and subsequent space ( ' ' ) character -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:padValue"
              as="xs:string">
  <xsl:param name="value" as="xs:string?"/>
  <xsl:value-of select="concat(' ', normalize-space($value), ' ')">
</xsl:function>
<!-- Returns the given $value with its values broken into tokens using whitespace as delimiters -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:tokenize"
              as="xs:string*">
  <xsl:param name="value" as="xs:string?"/>
  <xsl:sequence select="tokenize(normalize-space($value), ' ')">
</xsl:function>
<!-- Returns the given sequence of $values joined into a normalized single string -->
```

```
<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:join"
              as="xs:string">
  <xsl:param name="values" as="xs:string*" />
  <xsl:sequence select="normalize-space(string-join($values, ' '))" />
</xsl:function>
<!--
Returns true if any token in the attribute value matches at least one token in the provided list.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:containsAnyOfTheTokens"
              as="xs:boolean">
  <xsl:param name="attribute" />
  <xsl:param name="tokenList" as="xs:string*" />
  <xsl:value-of select="some $attrToken in tokenize(normalize-space(string($attribute)), ' ') satisfies $attrToken = $tokenList" />
</xsl:function>
<!-- Returns true if the given $relTo string (e.g. 'USA CAN GBR') contains any
tetragraphs that can be decomposed into its constituent countries -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:containsDecomposableTetra"
              as="xs:boolean">
  <xsl:param name="relTo" as="xs:string?" />
  <xsl:sequence select="normalize-space($relTo) and util:containsAnyOfTheTokens($relTo, $decomposableTetras)" />
</xsl:function>
<!-- Given a sequence of $relToStrings (e.g. ('USA CAN GBR', 'USA AUS SPAA')), returns a set of tokens
that are each of these $relToStrings decomposed using util:expandDecomposableTetras() -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:expandAllTetras"
              as="xs:string*">
  <xsl:param name="relToStrings" as="xs:string*" />
  <xsl:variable name="allTokens" as="xs:string*">
    <xsl:for-each select="$relToStrings">
      <xsl:variable name="expandedCountryTokens" select="util:expandDecomposableTetras(.)" />
      <xsl:value-of select="util:padValue(util:join($expandedCountryTokens))" />
    </xsl:for-each>
  </xsl:variable>
  <xsl:sequence select="$allTokens" />
</xsl:function>
<!-- Recursively remove all decomposable tetragraphs in the given $relTo string
and replace them with their constituent countries. Note: Does not include USA -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="util:expandDecomposableTetras"
              as="xs:string*">
  <xsl:param name="relTo" as="xs:string" />
  <xsl:variable name="expandedTetras">
    <xsl:choose>
      <xsl:when test="util:containsDecomposableTetra($relTo)">
        <xsl:variable name="currTetra"
                      select="util:tokenize($relTo)[. = $decomposableTetras][1]" />
        <xsl:variable name="currTetraCountries"
```

```

                select="util:join(util:getCountriesForTetra($currTetra))"/>
        <xsl:variable name="expandCurrTetra"
                select="replace(util:padValue($relTo), util:padValue($currTetra), util:padValue($currTetraCountries))"/>
        <xsl:value-of select="util:expandDecomposableTetras($expandCurrTetra)"/>
    </xsl:when>
    <xsl:otherwise>
        <xsl:value-of select="normalize-space($relTo)"/>
    </xsl:otherwise>
</xsl:choose>
</xsl:variable>
<xsl:sequence select="distinct-values(util:tokenize($expandedTetras))"/>
</xsl:function>
<!--*****-->
<!-- (U) Whitelist ID Rules -->
<!--*****-->
<!--(U) -->

<sch:include href="./Rules/WLAtomicEnergyMarkings_ID_00001.sch"/>
    <sch:include href="./Rules/WLClassification_ID_00001.sch"/>
    <sch:include href="./Rules/WLDisseminationControls_ID_00001.sch"/>
    <sch:include href="./Rules/WLFGISourceOpen_ID_00001.sch"/>
    <sch:include href="./Rules/WLFGISourceProtected_ID_00001.sch"/>
    <sch:include href="./Rules/WLISMAAttributes_ID_00001.sch"/>
    <sch:include href="./Rules/WLNtkAccessPolicy_ID_00001.sch"/>
    <sch:include href="./Rules/WLNtkAccessProfileValue_ID_00001.sch"/>
    <sch:include href="./Rules/WLNtkVocabulary_ID_00001.sch"/>
    <sch:include href="./Rules/WLReleaseableTo_ID_00001.sch"/>
    <sch:include href="./Rules/WLSARIdentifier_ID_00001.sch"/>
    <sch:include href="./Rules/WLSCIconcontrols_ID_00001.sch"/>
    <sch:include href="./Rules/WLdisplayOnlyTo_ID_00001.sch"/>
    <sch:include href="./Rules/WLnonICmarkings_ID_00001.sch"/>
    <sch:include href="./Rules/WLnonUSControls_ID_00001.sch"/>
    <sch:include href="./Rules/WLownerProducer_ID_00001.sch"/>
    <sch:include href="./Rules/WLNtkAccessPolicy_ID_00002.sch"/>
    <sch:include href="./Rules/WLNtkAccessProfileValue_ID_00002.sch"/>
    <sch:include href="./Rules/WLNtkVocabulary_ID_00002.sch"/>
    <sch:include href="./Rules/WLReleaseableTo_ID_00002.sch"/>
    <sch:include href="./Rules/WLownerProducer_ID_00002.sch"/>
    <sch:include href="./Rules/WLNtkAccessProfileValue_ID_00003.sch"/>
    <!--*****-->
<!-- (U) Whitelist Phases -->
<!--*****-->
</sch:schema>

<!--UNCLASSIFIED-->
```

Chapter 5 - Removed Rules

There are no rules that have been removed for *Whitelist*.