



Guide to Schematron Rules and Patterns

CEM Schematron Guide

Version 2018-APR

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/CEM_ID_00001.sch	3
2.2 - ../Rules/CEM_ID_00002.sch	4
2.3 - ../Rules/CEM_ID_00003.sch	5
2.4 - ../Rules/CEM_ID_00004.sch	6
2.5 - ../Rules/CEM_ID_00005.sch	7
Chapter 3 - Abstract Patterns	11
Chapter 4 - Schematron Schema	12
4.1 - ../CEM_XML.sch	13
Chapter 5 - Removed Rules	23

Chapter 1 - Introduction

1.1 - Purpose

This is an informative supplement for CEM. This guide is generated from the CEM 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 CEM. 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 CEM. The abstract patterns may be used in numbered rules or provided as reference for use in rules developed by users of CEM. Each abstract pattern has a code description and a code block with the abstract Schematron pattern.

Chapter 4 is a listing of the master CEM 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 CEM are encoded using ISO Schematron. Schematron is a rule-based validation language that uses XML Path Language to make assertions about an XML document.

CEM 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 CEM's Schematron rules.

Chapter 2 - Rules

All of the numbered Rules for CEM 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 CEM-ID-XXXXX, with rule files named CEM_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/CEM_ID_00001.sch

Rule Description

[CEM-ID-00001][Warning] DESVersion attributes SHOULD be specified as version 201804 with an optional extension.

Code Description

This rule supports extending the version identifier with an optional trailing hyphen and up to 23 additional characters. The version must match the regular expression “^201804(-.{1,23})?\$”.

Schematron Code

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

<sch:pattern id="CEM-ID-00001">
  <sch:rule id="CEM-ID-00001-R1" context="*[@cem:DESVersion]">
    <sch:assert test="matches(@cem:DESVersion, '^201804(-.{1,23})?$')"
      flag="warning"
      role="warning">[CEM-ID-00001][Warning] DESVersion attributes SHOULD be specified as version 201804 with an optional extension.</sch:assert>
  </sch:rule>
</sch:pattern>
```

2.2 - ../Rules/CEM_ID_00002.sch

Rule Description

[CEM-ID-00002][Error] For elements Facility and Person, if attribute xlink:href exists, then the attribute must have a non-null value. Human Readable: If attribute xlink:href exists for elements Facility and Person, it must have a value.

Code Description

This pattern uses an abstract rule to consolidate logic. It normalizes the space of the value of attribute xlink:href and makes sure the length of the resulting string is greater than zero, which indicates non-whitespace content. The abstract rule is extended once for each required element.

Schematron Code

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

<sch:pattern id="CEM-ID-00002"><!-- Abstract rule, which asserts that if attribute xlink:href exists, then it must have a non-null value -->

<sch:rule abstract="true" id="abs.rule00002">
    <sch:assert test="normalize-space(string(@xlink:href))"
               flag="error"
               role="error">[CEM-ID-00002][Error] For element
    <sch:name/>if attribute xlink:href exists, then the attribute must have a non-null value.
    </sch:assert>
</sch:rule>
<!-- Begin using abstract rule to check required elements -->

<sch:rule context="cem:Facility[@xlink:href]" id="CEM-ID-00002-R1">
    <sch:extends rule="abs.rule00002"/>
</sch:rule>
<sch:rule context="cem:Person[@xlink:href]" id="CEM-ID-00002-R2">
    <sch:extends rule="abs.rule00002"/>
</sch:rule>
</sch:pattern>
```

2.3 - ../Rules/CEM_ID_00003.sch

Rule Description

[CEM-ID-00003][Error] For attribute dateTimeRange, for each pair of date/time values, the second value must be later than the first value. Human Readable: The second value of date/time value pair for attribute dateTimeRange has to be later than the first value.

Code Description

The value of the attribute dateTimeRange is tokenized into a list of dateTimes, called \$dateTimeList. If the number of dates in \$dateTimeList is not even, then each date does not have a corresponding pair so this rule returns false. Otherwise, this rule verifies that the second date of each dateTime pair is later than the first date in that pair. To do this, the rule loops over all dates in \$dateTimeList and identify dateTime pairs by pairing the date at an even index N with the date at index N-1. For each pair, this rule verifies that \$dateList[N-1] is less than \$dateList[N].

Schematron Code

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

<sch:pattern id="CEM-ID-00003">
  <sch:rule context="@cem:dateTimeRange" id="CEM-ID-00003-R1">
    <sch:let name="dateTimeList" value="tokenize(string(@cem:dateTimeRange), ' ')" />
    <sch:assert test="if ((count($dateTimeList) mod 2) != 0) then false() else count( for $index in 1 to count($dateTimeList) return if($index mod 2 = 0) then
if(dtf:compareDateTimes($dateTimeList[$index - 1], '&lt;', $dateTimeList[$index])) then 1 else null else null ) = count($dateTimeList) * .5"
      flag="error"
      role="error">[CEM-ID-00003][Error] For attribute dateTimeRange, for each pair of date/time values, the second value must be later than the first value.
Human Readable: The second value of date/time value pair for attribute dateTimeRange has to be later than the first value.</sch:assert>
    </sch:rule>
  </sch:pattern>
```


2.4 - ../Rules/CEM_ID_00004.sch

Rule Description

[CEM-ID-00004][Error] If VIRT attributes are used on any CEM elements, then @virt:DESVersion must be declared.

Code Description

For any CEM elements with attribute virt:network, attribute virt:DESVersion must exist.

Schematron Code

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

<sch:pattern id="CEM-ID-00004">
    <sch:rule context="cem:*[@virt:*)" id="CEM-ID-00004-R1">
        <sch:assert test="@virt:DESVersion" flag="error" role="error">[CEM-ID-00004][Error] [CEM-ID-00004][Error] If VIRT attributes are used on any CEM elements, then
@virt:DESVersion must be declared.</sch:assert>
    </sch:rule>
</sch:pattern>
```

2.5 - ./Rules/CEM_ID_00005.sch

Rule Description

[CEM-ID-00005][Warning] For every optional element that exists and can have text content, the element should have non-null, non-whitespace value.

Code Description

This pattern uses an abstract rule to consolidate logic. The abstract rule first concatenates the text values within the given element, separated by a single space. The resultant string is then normalized with leading and trailing whitespace removed, and the length of the string is determined to be greater than zero, which indicates non-whitespace content. The abstract rule is extended once for each optional element in the CEM schema.

Schematron Code

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

<sch:pattern id="CEM-ID-00005">
    <sch:rule abstract="true" id="abs.rule00001">
        <sch:assert test="normalize-space(string())" flag="warning">[CEM-ID-00005][Warning] For every optional element that exists and can have text content, the element should
have non-null, non-whitespace value.</sch:assert>
    </sch:rule>
    <!-- Begin using abstract rule on optional elements -->
<!-- mixed='true' -->

<sch:rule context="cem:Drug">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<!-- Extensions of RunningTextType -->

<sch:rule context="cem:Account">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:CommData">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:CityName">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Commodity">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Concept">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:CountryName">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Date">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:DateTime">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:EntityUntyped">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Equipment">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Event">
    <sch:extends rule="abs.rule00001"/>
</sch:rule>
```

```
<sch:rule context="cem:Facility">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:GeoFeature">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:GeoRef">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Identifier">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:InfoBearer">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:LocationOfInterest">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:MilitaryUnit">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Money">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Nomenclature">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Organization">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Person">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:QuantityReference">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:SystemClass">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Term">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Time">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:TransportationNetwork">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Vehicle">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
<sch:rule context="cem:Weapon">
  <sch:extends rule="abs.rule00001"/>
</sch:rule>
```

</sch:pattern>

Chapter 3 - Abstract Patterns

There are no Abstract Patterns currently defined for CEM.

Chapter 4 - Schematron Schema

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

4.1 - `./CEM_XML.sch`

Code Description

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

Schematron Code

```
<!--UNCLASSIFIED-->
<?ICEA master?>
<!-- Notices - Distribution Notice:
    This document has been approved for Public Release and is available for use without restriction.
-->
<!-- WARNING:
    Once compiled into an XSLT the result will
    be the aggregate classification of all the CVES
    and included .sch files
-->

<sch:schema queryBinding="xslt2">
    <sch:ns uri="urn:us:gov:ic:cem" prefix="cem"/>
    <sch:ns uri="urn:us:gov:ic:virt" prefix="virt"/>
    <sch:ns uri="http://www.w3.org/1999/xlink" prefix="xlink"/>
    <sch:ns uri="date:time:function" prefix="dtf"/>
    <!-- ***** -->
<!-- * General Global Variables * -->
<!-- ***** -->

<sch:let name="timeZoneRegEx" value="'Z|[\+-]\d{2}:\d{2}'"/>
    <sch:let name="defaultTimeZone" value="'Z'"/>
    <sch:let name="startDateTimeTemplate" value="'0001-01-01T00:00:00.000'"/>
    <sch:let name="endDateTimeTemplate" value="'9999-12-01T23:59:59.999'"/>
    <!-- ***** -->
<!-- * Custom XSLT2 Function Definitions * -->
<!-- ***** -->
<!--
    Returns a string representation of the year portion of the date
    represented by the provided string.
    @param {xs:string} dateString String representation of a date in one
        of the allowable formats.
    @returns {xs:string} String representation of the year portion of the
        date represented by the provided string.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="dtf:getYear"
    as="xs:string">
    <xsl:param name="dateString" as="xs:string"/>
    <xsl:value-of select="substring(dtf:removeTimeZone($dateString), 1, 4)"/>
</xsl:function>
<!--
    Returns a string representation of the month portion of the date
    represented by the provided string.
    @param {xs:string} dateString String representation of a date in one
        of the allowable formats.
    @returns {xs:string} String representation of the month portion of the
        date represented by the provided string.
-->
```

```

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:getMonth"
              as="xs:string">
  <xsl:param name="dateString" as="xs:string"/>
  <xsl:value-of select="substring(dtf:removeTimeZone($dateString), 6, 2)"/>
</xsl:function>
<!--
@param {xs:date} date String representation of a date
@returns {xs:boolean} Returns true if the date provided occurs in a
    leap year; otherwise returns false.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:isLeapYear"
              as="xs:boolean">
  <xsl:param name="date" as="xs:string"/>
  <xsl:variable name="year" as="xs:integer" select="xs:integer(dtf:getYear($date))"/>
  <xsl:choose>
    <xsl:when test="$year mod 100 = 0">
      <xsl:choose>
        <xsl:when test="$year mod 400 = 0">
          <xsl:value-of select="true()"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="false()"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="$year mod 4 = 0">
          <xsl:value-of select="true()"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="false()"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:function>
<!--
Returns the maximum day of the month for an xs:dateTime as an xs:string.
@param {xs:dateTime} date The date time from which to get the month
@returns {xs:string} String representation of the maximum day of the month
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:getMaxDay"
              as="xs:string">
  <xsl:param name="date" as="xs:dateTime"/>
  <xsl:variable name="month" select="number(dtf:getMonth(string($date)))"/>
  <xsl:choose>
    <xsl:when test="$month = (1, 3, 5, 7, 8, 10, 12)">
      <xsl:value-of select="31"/>
    </xsl:when>
  </xsl:choose>

```

```

    </xsl:when>
    <xsl:when test="$month = (4, 6, 9, 11)">
      <xsl:value-of select="30"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="dtf:isLeapYear(string($date))">
          <xsl:value-of select="29"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="28"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:function>
<!--

```

Replaces the day portion of the provided dateTime with the new day provided.
 @param {xs:dateTime} dateTime An xs:dateTime to be updated with new day.
 @param {xs:string} newDayString String representation of day portion of a date.
 @returns {xs:dateTime} Returns new xs:dateTime with updated day portion.
 leap year; otherwise returns false.

```
-->
```

```

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:replaceDateTimeDay"
              as="xs:dateTime">
  <xsl:param name="dateTime" as="xs:dateTime"/>
  <xsl:param name="newDayString" as="xs:string"/>
  <xsl:variable name="beforeDay" select="substring(string($dateTime), 1, 8)"/>
  <xsl:variable name="afterDay" select="substring(string($dateTime), 11)"/>
  <xsl:value-of select="concat($beforeDay, $newDayString, $afterDay)"/>
</xsl:function>
<!--

```

Returns a string representation of the day portion of the date
 represented by the provided string.
 @param {xs:string} dateString String representation of a date in one
 of the allowable formats.
 @returns {xs:string} String representation of the day portion of the
 date represented by the provided string.

```
-->
```

```

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:getDay"
              as="xs:string">
  <xsl:param name="dateString" as="xs:string"/>
  <xsl:value-of select="substring(dtf:removeTimeZone($dateString), 9, 2)"/>
</xsl:function>
<!--

```

Removes the timezone portion of the date represented by the provided
 string and returns all remaining portions.
 @param {xs:string} dateString String representation of a date in one
 of the allowable formats.
 @returns {xs:string} String representation of a date without a timezone

```

    portion.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:removeTimeZone"
              as="xs:string">
    <xsl:param name="dateString" as="xs:string"/>
    <xsl:value-of select="replace($dateString, $timeZoneRegEx, '')"/>
</xsl:function>
<!--
    Uses the template provided to fill in missing portions of the string
    representation of a dateTime provided and returns a full xs:dateTime.
    The dateString provided must not contain a timezone.
    @param {xs:string} dateString String representation of a date in one
        of the allowable formats.
    @param {xs:string} dateTemplateString String template of a default date
        from which to pad missing portions of the dateString parameter.
    @returns {xs:dateTime} An xs:dateTime represented by the string date provided.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:padDateTimeWithTemplate"
              as="xs:dateTime">
    <xsl:param name="dateString" as="xs:string"/>
    <xsl:param name="dateTemplateString" as="xs:string"/>
    <xsl:value-of select="concat($dateString, substring($dateTemplateString, string-length(normalize-space($dateString)) + 1))"/>
</xsl:function>
<!--
    Returns a string representation of the timezone portion of the date
    represented by the provided string.
    @param {xs:string} dateString String representation of a date in one
        of the allowable formats.
    @returns {xs:string} String representation of the timezone portion of
        the date represented by the provided string.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:getTimeZone"
              as="xs:string">
    <xsl:param name="dateString" as="xs:string"/>
    <xsl:variable name="dateTimeEndingWithTimezone"
                  as="xs:string"
                  select="concat('^(, $timeZoneRegEx, ')$')"/>
    <xsl:choose>
        <xsl:when test="matches($dateString, $dateTimeEndingWithTimezone)">
            <xsl:value-of select="replace($dateString, $dateTimeEndingWithTimezone, '$1')"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="$defaultTimeZone"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:function>
<!--
    Calculates the date range implied for both primary and secondary and
```

determines if there is any overlap between the two ranges. Overlap is defined as the start of primary date range less than or equal to the end of secondary date range, inclusive, and the start of the secondary date range less than or equal to the end of the primary date range. Returns true if there is any overlap; otherwise, returns false.

@param {xs:string} primary String representation of a date in one of the allowable formats.

@param {xs:string} secondary String representation of a date in one of the allowable formats.

@returns {xs:boolean} Returns true if the date ranges implied by primary and secondary overlap at all; otherwise, returns false.

-->

```
<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:overlaps"
              as="xs:boolean">
  <xsl:param name="primary" as="xs:string"/>
  <xsl:param name="secondary" as="xs:string"/>
  <xsl:variable name="primaryStart"
                as="xs:dateTime"
                select="dtf:startDate($primary)"/>
  <xsl:variable name="primaryEnd" as="xs:dateTime" select="dtf:endDate($primary)"/>
  <xsl:variable name="secondaryStart"
                as="xs:dateTime"
                select="dtf:startDate($secondary)"/>
  <xsl:variable name="secondaryEnd"
                as="xs:dateTime"
                select="dtf:endDate($secondary)"/>
  <xsl:value-of select="$primaryStart &lt;= $secondaryEnd and $secondaryStart &lt;= $primaryEnd"/>
</xsl:function>
<!--
```

Determines if the date range implied by the string representation in primary is strictly before the date range implied by the string representation in secondary. Returns true if the end of the date range implied by primary is less than the start of the date range implied by secondary; otherwise, returns false.

@param {xs:string} primary String representation of a date in one of the allowable formats.

@param {xs:string} secondary String representation of a date in one of the allowable formats.

@returns {xs:boolean} Returns true if the date range implied by primary is strictly earlier than the date range implied by secondary; otherwise, returns false.

-->

```
<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              name="dtf:isBefore"
              as="xs:boolean">
  <xsl:param name="primary" as="xs:string"/>
  <xsl:param name="secondary" as="xs:string"/>
  <xsl:variable name="primaryEnd" as="xs:dateTime" select="dtf:endDate($primary)"/>
  <xsl:variable name="secondaryStart"
                as="xs:dateTime"
                select="dtf:startDate($secondary)"/>
```

```

        <xsl:value-of select="$primaryEnd &lt; $secondaryStart"/>
    </xsl:function>
    <!--
    Determines if the date range implied by the string representation in
    primary is strictly after the date range implied by the string
    representation in secondary. Returns true if the end of the date
    range implied by primary is less than the start of the date range
    implied by secondary; otherwise, returns false.
    @param {xs:string} primary String representation of a date in one
        of the allowable formats.
    @param {xs:string} secondary String representation of a date in one
        of the allowable formats.
    @returns {xs:boolean} Returns true if the date range implied by primary
        is stricly later than the date range implied by secondary; otherwise,
        returns false.
    -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="dtf:isAfter"
    as="xs:boolean">
    <xsl:param name="primary" as="xs:string"/>
    <xsl:param name="secondary" as="xs:string"/>
    <xsl:variable name="primaryStart"
        as="xs:dateTime"
        select="dtf:startDate($primary)"/>
    <xsl:variable name="secondaryEnd"
        as="xs:dateTime"
        select="dtf:endDate($secondary)"/>
    <xsl:value-of select="$secondaryEnd &lt; $primaryStart"/>
</xsl:function>
    <!--
    Returns the earlist xs:dateTime possible for the provided string
    representation of a dateTime. Fills in missing portions of the
    dateTime with the earliest possible values. Default values for missing
    portions:
    MM = 01
    DD = 01
    hh = 00
    mm = 00
    ss = 00
    s  = 000
    @param {xs:string} dateString String representation of a date in one
        of the allowable formats.
    @returns {xs:dateTime} The earliest xs:dateTime possible for the
        provided string representation of a dateTime.
    -->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="dtf:startDate"
    as="xs:dateTime">
    <xsl:param name="dateString" as="xs:string"/>
    <xsl:variable name="timeZonePortion" select="dtf:getTimeZone($dateString)"/>
    <xsl:variable name="dateTimePortion" select="dtf:removeTimeZone($dateString)"/>
    <xsl:variable name="outputDate"

```

```

        select="dtf:padDateTimeWithTemplate($dateTimePortion, $startDateTimeTemplate)"/>
    <xsl:value-of select="concat($outputDate, $timeZonePortion)"/>
</xsl:function>
<!--
Returns the latest xs:dateTime possible for the provided string
representation of a dateTime. Fills in missing portions of the
dateTime with the latest possible values. Default values for missing
portions:
MM = 12
DD = maximum day of the month
hh = 23
mm = 59
ss = 59
s  = 999
@param {xs:string} dateString String representation of a date in one
of the allowable formats.
@returns {xs:dateTime} The latest xs:dateTime possible for the
provided string representation of a dateTime.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="dtf:endDate"
    as="xs:dateTime">
    <xsl:param name="input" as="xs:string"/>
    <xsl:variable name="timeZonePortion" select="dtf:getTimeZone($input)"/>
    <xsl:variable name="dateTimePortion" select="dtf:removeTimeZone($input)"/>
    <xsl:variable name="outputDate"
        select="dtf:padDateTimeWithTemplate($dateTimePortion, $endDateTimeTemplate)"/>
    <xsl:variable name="outputWithCorrectedDay"
        select="dtf:replaceDateTimeDay($outputDate, dtf:getMaxDay($outputDate))"/>
    <xsl:choose>
        <xsl:when test="dtf:getDay($input)">
            <xsl:value-of select="concat($outputDate, $timeZonePortion)"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="concat($outputWithCorrectedDay, $timeZonePortion)"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:function>
<!--
Returns true if the year portion of the date represented by the provided
string contains four (4) digits; otherwise returns false.
@param {xs:string} dateString String representation of a date in one
of the allowable formats.
@returns {xs:string} true if the year portion of the date represented by
the provided string contains four (4) digits; otherwise returns false.
-->

<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="dtf:yearPortionHasFourDigits"
    as="xs:boolean">
    <xsl:param name="dateString" as="xs:string"/>
    <xsl:variable name="dateWithOnlyFourDigitYearAndOptionalTimeZoneRegEx"
        as="xs:string">
```



```

        select="concat('^\d{4}(', $timeZoneRegEx, ')?$')"/>
    <xsl:variable name="dateStartingWithFourDigitYearRegEx"
        as="xs:string"
        select="'^\d{4}-.*$'"/>
    <xsl:value-of select="matches($dateString, $dateWithOnlyFourDigitYearAndOptionalTimeZoneRegEx) or matches($dateString, $dateStartingWithFourDigitYearRegEx)"/>
</xsl:function>
<!--
Determines if the date range implied by the string representation in
primary satisfies the comparison to the date range implied by secondary
using the provided comparison operator; otherwise, returns false.

Both primary and secondary must be in one of the allowable formats
and represent dates with four digits in the year portion.
@param {xs:string} primary String representation of a date in one
of the allowable formats.
@param {xs:string} secondary String representation of a date in one
of the allowable formats.
@returns {xs:boolean} Returns true if the date range implied by primary
satisfies the comparison to the date range implied by secondary using
the provided comparison operator; otherwise, returns false.
-->
<xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    name="dtf:compareDateTimeRanges"
    as="xs:boolean">
    <xsl:param name="primary" as="xs:string"/>
    <xsl:param name="operator" as="xs:string"/>
    <xsl:param name="secondary" as="xs:string"/>
    <xsl:variable name="primaryAndSecondaryYearPortionsHaveFourDigits"
        as="xs:boolean"
        select="dtf:yearPortionHasFourDigits($primary) and dtf:yearPortionHasFourDigits($secondary)"/>
    <xsl:choose>
        <xsl:when test="$primaryAndSecondaryYearPortionsHaveFourDigits">
            <xsl:variable name="primaryStart"
                as="xs:dateTime"
                select="dtf:startDate($primary)"/>
            <xsl:variable name="primaryEnd" as="xs:dateTime" select="dtf:endDate($primary)"/>
            <xsl:variable name="secondaryStart"
                as="xs:dateTime"
                select="dtf:startDate($secondary)"/>
            <xsl:variable name="secondaryEnd"
                as="xs:dateTime"
                select="dtf:endDate($secondary)"/>
            <xsl:choose><!-- 'Less Than' Edge Case -->
                <xsl:when test="($operator = 'lt' or $operator = '&lt;') and (($primaryStart = $primaryEnd and $primaryStart = $secondaryStart) or ($primaryStart = $primaryEnd and $primaryStart = $secondaryEnd) or ($secondaryStart = $secondaryEnd and $primaryStart = $secondaryStart))">
                    <xsl:value-of select="false()"/>
                </xsl:when>
                <!-- 'Greater Than' Edge Case -->
                <xsl:when test="($operator = 'gt' or $operator = '&gt;') and (($primaryStart = $primaryEnd and $primaryEnd = $secondaryEnd) or ($primaryStart = $primaryEnd and $primaryEnd =

```



```
$secondaryStart) or ($secondaryStart = $secondaryEnd and $primaryEnd = $secondaryEnd))">
    <xsl:value-of select="false()"/>
</xsl:when>
<!-- 'Less Than' and 'Less Than or Equal' -->

<xsl:when test="$operator = 'lt' or $operator = '<';' or $operator = '<='">
    <xsl:value-of select="dtf:isBefore($primary, $secondary) or dtf:overlaps($primary, $secondary)"/>
</xsl:when>
<!-- 'Greater Than' and 'Greater Than or Equal' -->

<xsl:when test="$operator = 'gt' or $operator = '>';' or $operator = '>='">
    <xsl:value-of select="dtf:isAfter($primary, $secondary) or dtf:overlaps($primary, $secondary)"/>
</xsl:when>
<!-- Default to false() -->

<xsl:otherwise>
    <xsl:value-of select="false()"/>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
    <xsl:value-of select="false()"/>
</xsl:otherwise>
</xsl:choose>
</xsl:function>
<!--*****-->
<!-- (U) CEM ID Rules -->
<!--*****-->
<!--(U) -->

<sch:include href="./Rules/CEM_ID_00001.sch"/>
    <sch:include href="./Rules/CEM_ID_00002.sch"/>
    <sch:include href="./Rules/CEM_ID_00003.sch"/>
    <sch:include href="./Rules/CEM_ID_00004.sch"/>
    <sch:include href="./Rules/CEM_ID_00005.sch"/>
    <!--*****-->
<!-- (U) CEM Phases -->
<!--*****-->
</sch:schema>

<!--UNCLASSIFIED-->
```

Chapter 5 - Removed Rules

There are no rules that have been removed for CEM.