



Guide to XSLs for ISM-Rollup

ISM-Rollup XSL Guide

Version 2021-NOV

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
Chapter 2 - XSL Files	2
2.1 - functx-1.0-doc-2007-01.xsl	2
2.2 - ISM-Rollup.xsl	66
2.3 - ISM-Rollup-forXSpec.xsl	87

Chapter 1 - Introduction

1.1 - Purpose

This is an informative supplement for ISM-Rollup. This document provides XSL files developed for ISM-Rollup.

Chapter 2 - XSL Files

2.1 - functx-1.0-doc-2007-01.xsl

```
<!-- Commented out functions not used by Rollup. --><!--

*****
The FunctX XSLT Function Library
*****

Copyright (C) 2007 Datypic

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

For more information on the FunctX XSLT library, contact contrib@functx.com.

@version 1.0
@see http://www.xsltfunctions.com
--><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:dxmlf="http://www.datypic.com/xmlf"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:fn="http://www.w3.org/2005/xpath-functions"
                xmlns:local="http://www.datypic.com/local"
                xmlns:functx="http://www.functx.com"
                exclude-result-prefixes="dxmlf xs"
                version="2.0">

<!--
<!--\-
  Adds attributes to XML elements

@author Priscilla Walmsley, Datypic
@version 1.0
@see http://www.xsltfunctions.com/xsl/functx_add-attributes.html
@param $elements the element(s) to which you wish to add the attribute
@param $attrNames the name(s) of the attribute(s) to add
@param $attrValues the value(s) of the attribute(s) to add
-\->
<xsl:function name="functx:add-attributes" as="element()?"
            xmlns:functx="http://www.functx.com" >
  <xsl:param name="elements" as="element()*"/>
  <xsl:param name="attrNames" as="xs:QName*"/>
```

```
<xsl:param name="attrValues" as="xs:anyAtomicType*" />

<xsl:for-each select="$elements">
  <xsl:variable name="element" select="."/>
  <xsl:copy>
    <xsl:for-each select="$attrNames">
      <xsl:variable name="seq" select="position()" />
      <xsl:if test="not($element/@*[node-name(.) = current()])">
        <xsl:attribute name="{.}"
                      namespace="{namespace-uri-from-QName(.)}"
                      select="$attrValues[$seq]" />
      </xsl:if>
    </xsl:for-each>
    <xsl:copy-of select="@*/node()" />
  </xsl:copy>
</xsl:for-each>

</xsl:function>

<!--\--
  Adds months to a date

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_add-months.html
@param   $date the date
@param   $months the number of months to add
--\-->
<xsl:function name="functx:add-months" as="xs:date?"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />
  <xsl:param name="months" as="xs:integer" />

  <xsl:sequence select="
    xs:date($date) + functx:yearMonthDuration(0,$months)
  " />

</xsl:function>

<!--\--
  Adds attributes to XML elements

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_add-or-update-attributes.html
@param   $elements the element(s) to which you wish to add the attribute
@param   $attrNames the name(s) of the attribute(s) to add
@param   $attrValues the value(s) of the attribute(s) to add
--\-->
<xsl:function name="functx:add-or-update-attributes" as="element()?"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="elements" as="element()*" />
```

```
<xsl:param name="attrNames" as="xs:QName*" />
<xsl:param name="attrValues" as="xs:anyAtomicType*" />

<xsl:for-each select="$elements">
  <xsl:copy>
    <xsl:for-each select="$attrNames">
      <xsl:variable name="seq" select="position()" />
      <xsl:attribute name="{.}"
                    namespace="{namespace-uri-from-QName(.)}"
                    select="$attrValues[$seq]" />
    </xsl:for-each>
    <xsl:copy-of select="*[not(node-name(.) = $attrNames)]|node()" />
  </xsl:copy>
</xsl:for-each>

</xsl:function>

<!--\--
  Whether a value is all whitespace or a zero-length string

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_all-whitespace.html
@param   $arg the string (or node) to test
--\-->
<xsl:function name="functx:all-whitespace" as="xs:boolean"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    normalize-space($arg) = ''
  " />

</xsl:function>

<!--\--
  Whether all the values in a sequence are distinct

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_are-distinct-values.html
@param   $seq the sequence of values
--\-->
<xsl:function name="functx:are-distinct-values" as="xs:boolean"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    count(distinct-values($seq)) = count($seq)
  " />

</xsl:function>
```

```
<!--\-
  The built-in type of an atomic value

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_atomic-type.html
@param   $values the value(s) whose type you want to determine
--\->
<xsl:function name="functx:atomic-type" as="xs:string*"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="values" as="xs:anyAtomicType*" />

  <xsl:sequence select="
for $val in $values
return
(if ($val instance of xs:untypedAtomic) then 'xs:untypedAtomic'
else if ($val instance of xs:anyURI) then 'xs:anyURI'
else if ($val instance of xs:string) then 'xs:string'
else if ($val instance of xs:QName) then 'xs:QName'
else if ($val instance of xs:boolean) then 'xs:boolean'
else if ($val instance of xs:base64Binary) then 'xs:base64Binary'
else if ($val instance of xs:hexBinary) then 'xs:hexBinary'
else if ($val instance of xs:integer) then 'xs:integer'
else if ($val instance of xs:decimal) then 'xs:decimal'
else if ($val instance of xs:float) then 'xs:float'
else if ($val instance of xs:double) then 'xs:double'
else if ($val instance of xs:date) then 'xs:date'
else if ($val instance of xs:time) then 'xs:time'
else if ($val instance of xs:dateTime) then 'xs:dateTime'
else if ($val instance of xs:dayTimeDuration)
  then 'xs:dayTimeDuration'
else if ($val instance of xs:yearMonthDuration)
  then 'xs:yearMonthDuration'
else if ($val instance of xs:duration) then 'xs:duration'
else if ($val instance of xs:gMonth) then 'xs:gMonth'
else if ($val instance of xs:gYear) then 'xs:gYear'
else if ($val instance of xs:gYearMonth) then 'xs:gYearMonth'
else if ($val instance of xs:gDay) then 'xs:gDay'
else if ($val instance of xs:gMonthDay) then 'xs:gMonthDay'
else 'unknown')
"/>

</xsl:function>

<!--\-
  The average, counting "empty" values as zero

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_avg-empty-is-zero.html
@param   $values the values to be averaged
```



```
@param  $allNodes the sequence of all nodes to find the average over
-|->
<xsl:function name="functx:avg-empty-is-zero" as="xs:double"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="values" as="xs:anyAtomicType*" />
    <xsl:param name="allNodes" as="node(*)" />

    <xsl:sequence select="
        if (empty($allNodes))
        then 0
        else sum($values[string(.) != '']) div count($allNodes)
    "/>

</xsl:function>

<!--\--
    Whether a value is between two provided values

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_between-exclusive.html
@param   $value the value to be tested
@param   $minValue the minimum value
@param   $maxValue the maximum value
-|->
<xsl:function name="functx:between-exclusive" as="xs:boolean"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="value" as="xs:anyAtomicType?" />
    <xsl:param name="minValue" as="xs:anyAtomicType" />
    <xsl:param name="maxValue" as="xs:anyAtomicType" />

    <xsl:sequence select="
        $value > $minValue and $value < $maxValue
    "/>

</xsl:function>

<!--\--
    Whether a value is between two provided values, or equal to one of them

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_between-inclusive.html
@param   $value the value to be tested
@param   $minValue the minimum value
@param   $maxValue the maximum value
-|->
<xsl:function name="functx:between-inclusive" as="xs:boolean"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="value" as="xs:anyAtomicType?" />
    <xsl:param name="minValue" as="xs:anyAtomicType" />
    <xsl:param name="maxValue" as="xs:anyAtomicType" />
```

```
<xsl:sequence select="
    $value >= $minValue and $value <= $maxValue
"/>

</xsl:function>

<!--
    Turns a camelCase string into space-separated words

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_camel-case-to-words.html
    @param    $arg the string to modify
    @param    $delim the delimiter for the words (e.g. a space)
-->
<xsl:function name="functx:camel-case-to-words" as="xs:string"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?" />
    <xsl:param name="delim" as="xs:string" />

    <xsl:sequence select="
        concat(substring($arg,1,1),
            replace(substring($arg,2),'\p{Lu}',' '),
            concat($delim, '$1'))
    "/>

</xsl:function>

<!--
    Capitalizes the first character of a string

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_capitalize-first.html
    @param    $arg the word or phrase to capitalize
-->
<xsl:function name="functx:capitalize-first" as="xs:string?"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?" />

    <xsl:sequence select="
        concat(upper-case(substring($arg,1,1)),
            substring($arg,2))
    "/>

</xsl:function>

<!--
    Changes the names of elements in an XML fragment
```

```
@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_change-element-names-deep.html
@param   $nodes the element(s) to change
@param   $oldNames the sequence of names to change from
@param   $newNames the sequence of names to change to
-|->
<xsl:function name="functx:change-element-names-deep" as="node()*"
      xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node()*"/>
  <xsl:param name="oldNames" as="xs:QName*" />
  <xsl:param name="newNames" as="xs:QName*" />

  <xsl:if test="count($oldNames) != count($newNames)">
    <xsl:sequence select="error(
      xs:QName('functx:Different_number_of_names'))"/>
  </xsl:if>
  <xsl:for-each select="$nodes">
    <xsl:variable name="node" select="."/>
    <xsl:choose>
      <xsl:when test="$node instance of element()">
        <xsl:variable name="theName"
          select="functx:if-empty
            ($newNames[index-of($oldNames, node-name($node))],
            node-name($node))"/>
        <xsl:element name="{ $theName }"
          namespace="{ namespace-uri-from-QName( $theName ) }" >
          <xsl:sequence select="($node/@*,
            functx:change-element-names-deep($node/node(),
              $oldNames, $newNames))"/>
        </xsl:element>
      </xsl:when>
      <xsl:when test="$node instance of document-node()">
        <xsl:document>
          <xsl:sequence select="functx:change-element-names-deep(
            $node/node(), $oldNames, $newNames)"/>
        </xsl:document>
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="$node"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:function>

<!--\--
  Changes the namespace of XML elements and its descendants

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_change-element-ns-deep.html
  @param   $nodes the nodes to change
```

```
@param    $newns the new namespace
@param    $prefix the prefix to use for the new namespace
-|->
<xsl:function name="functx:change-element-ns-deep" as="node(*)"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node(*)"/>
  <xsl:param name="newns" as="xs:string"/>
  <xsl:param name="prefix" as="xs:string"/>

  <xsl:for-each select="$nodes">
    <xsl:variable name="node" select="."/>
    <xsl:choose>
      <xsl:when test="$node instance of element()">
        <xsl:element name="{concat($prefix,
                                   if ($prefix = '')
                                     then ''
                                     else ': ',
                                   local-name($node))}"
                    namespace="{ $newns }">
          <xsl:sequence select="($node/@*,
                               functx:change-element-ns-deep($node/node(),
                                                                $newns, $prefix))"/>
        </xsl:element>
      </xsl:when>
      <xsl:when test="$node instance of document-node()">
        <xsl:document>
          <xsl:sequence select="functx:change-element-ns-deep(
                                $node/node(), $newns, $prefix)"/>
        </xsl:document>
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="$node"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:function>

<!--
  Changes the namespace of XML elements

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_change-element-ns.html
  @param   $elements the elements to change
  @param   $newns the new namespace
  @param   $prefix the prefix to use for the new namespace
-|->
<xsl:function name="functx:change-element-ns" as="element()?"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="elements" as="element(*)"/>
  <xsl:param name="newns" as="xs:string"/>
  <xsl:param name="prefix" as="xs:string"/>
```

```
<xsl:for-each select="$elements">
  <xsl:variable name="element" select="."/>
  <xsl:element name="{concat($prefix,
                            if ($prefix = '')
                              then ''
                              else ': ',
                            local-name($element))}"
              namespace="{ $newsns }">
    <xsl:sequence select="$element/@*, $element/node()" />
  </xsl:element>
</xsl:for-each>
```

```
</xsl:function>
```

```
<!--
  Converts a string to a sequence of characters
```

```
@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_chars.html
@param   $arg the string to split
```

```
-->
<xsl:function name="functx:chars" as="xs:string*"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
```

```

  <xsl:sequence select="
    for $ch in string-to-codepoints($arg)
    return codepoints-to-string($ch)
  " />
```

```
</xsl:function>
```

```
<!--
  Whether a string contains any of a sequence of strings
```

```
@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_contains-any-of.html
@param   $arg the string to test
@param   $searchStrings the strings to look for
```

```
-->
<xsl:function name="functx:contains-any-of" as="xs:boolean"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="searchStrings" as="xs:string*" />
```

```

  <xsl:sequence select="
    some $searchString in $searchStrings
    satisfies contains($arg,$searchString)
  " />
```

```
</xsl:function>

<!--
  Whether one string contains another, without regard to case

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_contains-case-insensitive.html
  @param   $arg the string to search
  @param   $substring the substring to find
-->
<xsl:function name="functx:contains-case-insensitive" as="xs:boolean?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?"/>
  <xsl:param name="substring" as="xs:string"/>

  <xsl:sequence select="
    contains(upper-case($arg), upper-case($substring))
  "/>
</xsl:function>

<!--
  Whether one string contains another, as a separate word

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_contains-word.html
  @param   $arg the string to search
  @param   $word the word to find
-->
<xsl:function name="functx:contains-word" as="xs:boolean"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?"/>
  <xsl:param name="word" as="xs:string"/>

  <xsl:sequence select="
    matches(upper-case($arg),
      concat('^(.*\W)?',
        upper-case(functx:escape-for-regex($word)),
        '(\W.*)?$',
      )
  "/>
</xsl:function>

<!--
  Copies attributes from one element to another

  @author  Priscilla Walmsley, Datypic
  @version 1.0
```

```
@see      http://www.xsltfunctions.com/xsl/functx_copy-attributes.html
@param    $copyTo the element to copy attributes to
@param    $copyFrom the element to copy attributes from
-|->
<xsl:function name="functx:copy-attributes" as="element()"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="copyTo" as="element()"/>
  <xsl:param name="copyFrom" as="element()"/>

  <xsl:element name="{node-name($copyTo)}">
    <xsl:sequence select="
      ($copyTo/@*[not(node-name(.) = $copyFrom/@*/node-name(.))],
      $copyFrom/@*,
      $copyTo/node())"/>
  </xsl:element>

</xsl:function>

<!--
  Construct a date from a year, month and day

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_date.html
@param    $year the year
@param    $month the month
@param    $day the day
-|->
<xsl:function name="functx:date" as="xs:date"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="year" as="xs:anyAtomicType"/>
  <xsl:param name="month" as="xs:anyAtomicType"/>
  <xsl:param name="day" as="xs:anyAtomicType"/>

  <xsl:sequence select="
    xs:date(
      concat(
        functx:pad-integer-to-length(xs:integer($year),4),'-',
        functx:pad-integer-to-length(xs:integer($month),2),'-',
        functx:pad-integer-to-length(xs:integer($day),2)))
  "/>

</xsl:function>

<!--
  Construct a date/time from individual components

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_datetime.html
@param    $year the year
@param    $month the month
```

```
@param    $day the day
@param    $hour the hour
@param    $minute the minute
@param    $second the second
-|->
<xsl:function name="functx:dateTime" as="xs:dateTime"
            xmlns:functx="http://www.functx.com" >
    <xsl:param name="year" as="xs:anyAtomicType"/>
    <xsl:param name="month" as="xs:anyAtomicType"/>
    <xsl:param name="day" as="xs:anyAtomicType"/>
    <xsl:param name="hour" as="xs:anyAtomicType"/>
    <xsl:param name="minute" as="xs:anyAtomicType"/>
    <xsl:param name="second" as="xs:anyAtomicType"/>

    <xsl:sequence select="
        xs:dateTime(
            concat(functx:date($year,$month,$day),'T',
                functx:time($hour,$minute,$second))
        "/>
</xsl:function>

<!--\--
    The day of the year (a number between 1 and 366)

    @author    Priscilla Walmsley, Datypic
    @version    1.0
    @see        http://www.xsltfunctions.com/xsl/functx_day-in-year.html
    @param    $date the date
-|->
<xsl:function name="functx:day-in-year" as="xs:integer?"
            xmlns:functx="http://www.functx.com" >
    <xsl:param name="date" as="xs:anyAtomicType?"/>

    <xsl:sequence select="
        days-from-duration(
            xs:date($date) - functx:first-day-of-year($date)) + 1
        "/>
</xsl:function>

<!--\--
    The abbreviated day of the week, from a date, in English

    @author    Priscilla Walmsley, Datypic
    @version    1.0
    @see        http://www.xsltfunctions.com/xsl/functx_day-of-week-abbrev-en.html
    @param    $date the date
-|->
<xsl:function name="functx:day-of-week-abbrev-en" as="xs:string?"
            xmlns:functx="http://www.functx.com" >
    <xsl:param name="date" as="xs:anyAtomicType?"/>
```



```
<xsl:sequence select="
  ('Sun', 'Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat')
  [functx:day-of-week($date) + 1]
"/>

</xsl:function>

<!--\--
  The name of the day of the week, from a date, in English

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_day-of-week-name-en.html
  @param   $date the date
--\-->
<xsl:function name="functx:day-of-week-name-en" as="xs:string?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    ('Sunday', 'Monday', 'Tuesday', 'Wednesday',
     'Thursday', 'Friday', 'Saturday')
    [functx:day-of-week($date) + 1]
  "/>

</xsl:function>

<!--\--
  The day of the week, from a date

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_day-of-week.html
  @param   $date the date
--\-->
<xsl:function name="functx:day-of-week" as="xs:integer?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    if (empty($date))
    then ()
    else xs:integer((xs:date($date) - xs:date('1901-01-06'))
      div xs:dayTimeDuration('P1D')) mod 7
  "/>

</xsl:function>

<!--\--
  Construct a dayTimeDuration from a number of days, hours, etc.
```

```
@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_daytimeduration.html
@param   $days the number of days
@param   $hours the number of hours
@param   $minutes the number of minutes
@param   $seconds the number of seconds
- \->
<xsl:function name="functx:dayTimeDuration" as="xs:dayTimeDuration"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="days" as="xs:decimal?"/>
  <xsl:param name="hours" as="xs:decimal?"/>
  <xsl:param name="minutes" as="xs:decimal?"/>
  <xsl:param name="seconds" as="xs:decimal?"/>

  <xsl:sequence select="
    (xs:dayTimeDuration('P1D') * functx:if-empty($days,0)) +
    (xs:dayTimeDuration('PT1H') * functx:if-empty($hours,0)) +
    (xs:dayTimeDuration('PT1M') * functx:if-empty($minutes,0)) +
    (xs:dayTimeDuration('PT1S') * functx:if-empty($seconds,0))
  "/>
</xsl:function>

<!--\--
  Number of days in the month

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_days-in-month.html
  @param   $date the date
- \->
<xsl:function name="functx:days-in-month" as="xs:integer?"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?"/>

  <xsl:sequence select="
    if (month-from-date(xs:date($date)) = 2 and
        functx:is-leap-year($date))
    then 29
    else
    (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
    [month-from-date(xs:date($date))]
```

```
  "/>
</xsl:function>

<!--\--
  The depth (level) of a node in an XML tree

  @author  Priscilla Walmsley, Datypic
```

```
@version 1.0
@see      http://www.xsltfunctions.com/xsl/functx_depth-of-node.html
@param    $node the node to check
-|->
<xsl:function name="functx:depth-of-node" as="xs:integer"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="node" as="node()?" />

  <xsl:sequence select="
    count($node/ancestor-or-self::node())
  " />
</xsl:function>

<!--
  The distinct names of all attributes in an XML fragment

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see      http://www.xsltfunctions.com/xsl/functx_distinct-attribute-names.html
  @param    $nodes the root to start from
-|->
<xsl:function name="functx:distinct-attribute-names" as="xs:string*"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node(*)" />

  <xsl:sequence select="
    distinct-values($nodes//@*/name())
  " />
</xsl:function>
-->

<!--
  The XML nodes with distinct values, taking into account attributes and descendants

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see      http://www.xsltfunctions.com/xsl/functx_distinct-deep.html
  @param    $nodes the sequence of nodes to test
-->
<xsl:function name="functx:distinct-deep" as="node(*)">
  <xsl:param name="nodes" as="node(*)" />

  <xsl:sequence select="
    for $seq in (1 to count($nodes))
    return $nodes[$seq][not(functx:is-node-in-sequence-deep-equal(
$nodes[position() < $seq]))] " />

  </xsl:function>

  <!--

  <!--\--
  The distinct names of all elements in an XML fragment
```

```
@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_distinct-element-names.html
@param   $nodes the root(s) to start from
-|->
<xsl:function name="functx:distinct-element-names" as="xs:string*"
            xmlns:functx="http://www.functx.com" >
    <xsl:param name="nodes" as="node(*)*" />

    <xsl:sequence select="
        distinct-values($nodes/descendant-or-self::* /name(.))
    " />

</xsl:function>

<!--\--
    The distinct paths of all descendant elements in an XML fragment

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_distinct-element-paths.html
@param   $nodes the root(s) to start from
-|->
<xsl:function name="functx:distinct-element-paths" as="xs:string*"
            xmlns:functx="http://www.functx.com" >
    <xsl:param name="nodes" as="node(*)*" />

    <xsl:sequence select="
        distinct-values(functx:path-to-node($nodes/descendant-or-self::*))
    " />

</xsl:function>

<!--\--
    The distinct XML nodes in a sequence (by node identity)

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_distinct-nodes.html
@param   $nodes the node sequence
-|->
<xsl:function name="functx:distinct-nodes" as="node(*)*"
            xmlns:functx="http://www.functx.com" >
    <xsl:param name="nodes" as="node(*)*" />

    <xsl:sequence select="
        for $seq in (1 to count($nodes))
        return $nodes[$seq][not(functx:is-node-in-sequence(
            .,$nodes[position() < $seq]))]
    " />

</xsl:function>
```

```
<!--\-
  Converts a timezone like "-05:00" or "Z" into xs:dayTimeDuration

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_duration-from-timezone.html
@param   $timezone the time zone, in (+|-)HH:MM format
--\->
<xsl:function name="functx:duration-from-timezone" as="xs:dayTimeDuration"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="timezone" as="xs:string"/>

  <xsl:sequence select="
    xs:dayTimeDuration(
      if (not(matches($timezone,'Z|[\+|-]\d{2}:\d{2}'))
      then error(xs:QName('functx:Invalid_Timezone_Value'))
      else if ($timezone = 'Z')
      then 'PT0S'
      else replace($timezone,'\\+?(\\d{2}):\\d{2}','PT$1H')
      )
    "/>
</xsl:function>

<!--\-
  Dynamically evaluates a simple XPath path

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_dynamic-path.html
@param   $parent the root to start from
@param   $path the path expression
--\->
<xsl:function name="functx:dynamic-path" as="item()*"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="parent" as="node()"/>
  <xsl:param name="path" as="xs:string"/>

  <xsl:variable name="nextStep"
    select="functx:substring-before-if-contains($path,'/')"/>
  <xsl:variable name="restOfSteps"
    select="substring-after($path,'/')"/>
  <xsl:for-each select="
    ($parent/*[functx:name-test(name(),$nextStep)],
    $parent/@*[functx:name-test(name(),
                                substring-after($nextStep,'@'))])">
    <xsl:variable name="child" select="."/>
    <xsl:sequence select="if ($restOfSteps)
      then functx:dynamic-path($child, $restOfSteps)
      else $child"/>
  </xsl:for-each>
```

```
</xsl:function>

<!--
  Escapes regex special characters

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_escape-for-regex.html
  @param   $arg the string to escape
-->
<xsl:function name="functx:escape-for-regex" as="xs:string"
             xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    replace($arg,
      '(\.|\[|\]|\\|\/|\-|\^|\$|\?|\*|\+|\{|\}|\(|\))', '\\\$1')
  " />
</xsl:function>

<!--
  Whether one (and only one) of two boolean values is true

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_exclusive-or.html
  @param   $arg1 the first boolean value
  @param   $arg2 the second boolean value
-->
<xsl:function name="functx:exclusive-or" as="xs:boolean?"
             xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg1" as="xs:boolean?" />
  <xsl:param name="arg2" as="xs:boolean?" />

  <xsl:sequence select="
    $arg1 != $arg2
  " />
</xsl:function>

<!--
  The first day of the month of a date

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_first-day-of-month.html
  @param   $date the date
-->
<xsl:function name="functx:first-day-of-month" as="xs:date?"
```

```

        xmlns:functx="http://www.functx.com" >
<xsl:param name="date" as="xs:anyAtomicType?" />

<xsl:sequence select="
    functx:date(year-from-date(xs:date($date)),
        month-from-date(xs:date($date)),
        1)
"/>

</xsl:function>

<!--
    The first day of the year of a date

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_first-day-of-year.html
    @param    $date the date
-->
<xsl:function name="functx:first-day-of-year" as="xs:date?"
        xmlns:functx="http://www.functx.com" >
    <xsl:param name="date" as="xs:anyAtomicType?" />

    <xsl:sequence select="
        functx:date(year-from-date(xs:date($date)), 1, 1)
    "/>

</xsl:function>

<!--
    The XML node in a sequence that appears first in document order

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_first-node.html
    @param    $nodes the sequence of nodes
-->
<xsl:function name="functx:first-node" as="node()?"
        xmlns:functx="http://www.functx.com" >
    <xsl:param name="nodes" as="node(*)" />

    <xsl:sequence select="
        ($nodes/.)[1]
    "/>

</xsl:function>

<!--
    Whether an XML node follows another without being its descendant

    @author   W3C XML Query Working Group
```

```
@version 1.0
@see      http://www.xsltfunctions.com/xsl/functx_follows-not-descendant.html
@param    $a the first node
@param    $b the second node
-|->
<xsl:function name="functx:follows-not-descendant" as="xs:boolean"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="a" as="node()?" />
  <xsl:param name="b" as="node()?" />

  <xsl:sequence select="
    $a >> $b and empty($b intersect $a/ancestor::node())
  "/>

</xsl:function>

<!--
  Moves title words like "the" and "a" to the end of strings

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_format-as-title-en.html
@param    $titles the titles to format
-|->
<xsl:function name="functx:format-as-title-en" as="xs:string*"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="titles" as="xs:string*" />

  <xsl:variable name="wordsToMoveToEnd"
                select="('A', 'An', 'The')"/>
  <xsl:for-each select="$titles">
    <xsl:variable name="title" select="."/>
    <xsl:variable name="firstWord"
                  select="functx:substring-before-match($title,'\W')"/>
    <xsl:sequence select="if ($firstWord = $wordsToMoveToEnd)
      then replace($title,'(.*?)\W(.*?)', '$2, $1')
      else $title"/>
  </xsl:for-each>

</xsl:function>

<!--
  Returns the fragment from a URI

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_fragment-from-uri.html
@param    $uri the URI
-|->
<xsl:function name="functx:fragment-from-uri" as="xs:string?"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="uri" as="xs:string?" />
```



```
<xsl:sequence select="
  substring-after($uri,'#')
"/>

</xsl:function>

<!--\-
  Whether an element has element-only content

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_has-element-only-content.html
@param    $element the XML element to test
-\->
<xsl:function name="functx:has-element-only-content" as="xs:boolean"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="element" as="element()"/>

  <xsl:sequence select="
    not($element/text()[normalize-space(.) != '']) and $element/*
  "/>

</xsl:function>

<!--\-
  Whether an element has empty content

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_has-empty-content.html
@param    $element the XML element to test
-\->
<xsl:function name="functx:has-empty-content" as="xs:boolean"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="element" as="element()"/>

  <xsl:sequence select="
    not($element/node())
  "/>

</xsl:function>

<!--\-
  Whether an element has mixed content

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_has-mixed-content.html
@param    $element the XML element to test
-\->
```

```
<xsl:function name="functx:has-mixed-content" as="xs:boolean"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="element" as="element()"/>

    <xsl:sequence select="
        $element/text()[normalize-space(.) != ''] and $element/*
    "/>

</xsl:function>

<!--\--
    Whether an element has simple content

    @author    Priscilla Walmsley, Datypic
    @version 1.0
    @see       http://www.xsltfunctions.com/xsl/functx_has-simple-content.html
    @param     $element the XML element to test
--\-->
<xsl:function name="functx:has-simple-content" as="xs:boolean"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="element" as="element()"/>

    <xsl:sequence select="
        $element/text() and not($element/*)
    "/>

</xsl:function>

<!--\--
    Gets the ID of an XML element

    @author    Priscilla Walmsley, Datypic
    @version 1.0
    @see       http://www.xsltfunctions.com/xsl/functx_id-from-element.html
    @param     $element the element
--\-->
<xsl:function name="functx:id-from-element" as="xs:string?"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="element" as="element()?" />

    <xsl:sequence select="
        data(($element/@*[id(.) is ..])[1])
    "/>

</xsl:function>

<!--\--
    Gets XML element(s) that have an attribute with a particular value

    @author    Priscilla Walmsley, Datypic
    @version 1.0
```

```
@see      http://www.xsltfunctions.com/xsl/functx_id-untyped.html
@param    $node the root node(s) to start from
@param    $id the "id" to find
-|->
<xsl:function name="functx:id-untyped" as="element(*)"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="node" as="node(*)"/>
  <xsl:param name="id" as="xs:anyAtomicType"/>

  <xsl:sequence select="
    $node//*[ @* = $id ]
  " />

</xsl:function>

<!--
  The first argument if it is not empty, otherwise the second argument

@author   W3C XML Query WG
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_if-absent.html
@param    $arg the item(s) that may be absent
@param    $value the item(s) to use if the item is absent
-|->
<xsl:function name="functx:if-absent" as="item(*)"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="item(*)"/>
  <xsl:param name="value" as="item(*)"/>

  <xsl:sequence select="
    if (exists($arg))
      then $arg
      else $value
  " />

</xsl:function>

<!--
  The first argument if it is not blank, otherwise the second argument

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_if-empty.html
@param    $arg the node that may be empty
@param    $value the item(s) to use if the node is empty
-|->
<xsl:function name="functx:if-empty" as="item(*)"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="item()?"/>
  <xsl:param name="value" as="item(*)"/>

  <xsl:sequence select="
```

```

    if (string($arg) != '')
    then data($arg)
    else $value
"/>

</xsl:function>

<!--\--
    The position of a node in a sequence, based on contents and attributes

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_index-of-deep-equal-node.html
    @param    $nodes the node sequence
    @param    $nodeToFind the node to find in the sequence
--\-->
<xsl:function name="functx:index-of-deep-equal-node" as="xs:integer*"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="nodes" as="node()*"/>
    <xsl:param name="nodeToFind" as="node()"/>

    <xsl:sequence select="
    for $seq in (1 to count($nodes))
    return $seq[deep-equal($nodes[$seq],$nodeToFind)]
    "/>

</xsl:function>

<!--\--
    The first position of a matching substring

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_index-of-match-first.html
    @param    $arg the string
    @param    $pattern the pattern to match
--\-->
<xsl:function name="functx:index-of-match-first" as="xs:integer?"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?"/>
    <xsl:param name="pattern" as="xs:string"/>

    <xsl:sequence select="
    if (matches($arg,$pattern))
    then string-length(tokenize($arg, $pattern)[1]) + 1
    else ()
    "/>

</xsl:function>

<!--\--
```

```

    The position of a node in a sequence, based on node identity

@author   W3C XML Query Working Group
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_index-of-node.html
@param    $nodes the node sequence
@param    $nodeToFind the node to find in the sequence
-|->
<xsl:function name="functx:index-of-node" as="xs:integer*"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="nodes" as="node()*"/>
    <xsl:param name="nodeToFind" as="node()"/>

    <xsl:sequence select="
    for $seq in (1 to count($nodes))
    return $seq[$nodes[$seq] is $nodeToFind]
    "/>

</xsl:function>

<!--
    The first position of a substring

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_index-of-string-first.html
@param    $arg the string
@param    $substring the substring to find
-|->
<xsl:function name="functx:index-of-string-first" as="xs:integer?"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?"/>
    <xsl:param name="substring" as="xs:string"/>

    <xsl:sequence select="
    if (contains($arg, $substring))
    then string-length(substring-before($arg, $substring))+1
    else ()
    "/>

</xsl:function>

<!--
    The last position of a substring

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_index-of-string-last.html
@param    $arg the string
@param    $substring the substring to find
-|->
<xsl:function name="functx:index-of-string-last" as="xs:integer?"
```

```

        xmlns:functx="http://www.functx.com" >
<xsl:param name="arg" as="xs:string?"/>
<xsl:param name="substring" as="xs:string"/>

    <xsl:sequence select="
    functx:index-of-string($arg, $substring)[last()]
    "/>

</xsl:function>

<!--\--
    The position(s) of a substring

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_index-of-string.html
    @param    $arg the string
    @param    $substring the substring to find
--\-->
<xsl:function name="functx:index-of-string" as="xs:integer*"
        xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?"/>
    <xsl:param name="substring" as="xs:string"/>

    <xsl:sequence select="
    if (contains($arg, $substring))
    then (string-length(substring-before($arg, $substring))+1,
        for $other in
            functx:index-of-string(substring-after($arg, $substring),
                $substring)

        return
            $other +
            string-length(substring-before($arg, $substring)) +
            string-length($substring)

    else ()
    "/>

</xsl:function>

<!--\--
    Inserts a string at a specified position

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_insert-string.html
    @param    $originalString the original string to insert into
    @param    $stringToInsert the string to insert
    @param    $pos the position
--\-->
<xsl:function name="functx:insert-string" as="xs:string"
        xmlns:functx="http://www.functx.com" >
    <xsl:param name="originalString" as="xs:string?"/>
```

```
<xsl:param name="stringToInsert" as="xs:string?"/>
<xsl:param name="pos" as="xs:integer"/>

<xsl:sequence select="
  concat(substring($originalString,1,$pos - 1),
    $stringToInsert,
    substring($originalString,$pos))
"/>

</xsl:function>

<!--
  Whether a value is numeric

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_is-a-number.html
  @param   $value the value to test
-->
<xsl:function name="functx:is-a-number" as="xs:boolean"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="value" as="xs:anyAtomicType?"/>

  <xsl:sequence select="
    string(number($value)) != 'NaN'
  "/>

</xsl:function>

<!--
  Whether a URI is absolute

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_is-absolute-uri.html
  @param   $uri the URI to test
-->
<xsl:function name="functx:is-absolute-uri" as="xs:boolean"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="uri" as="xs:string?"/>

  <xsl:sequence select="
    matches($uri,'^[a-z]+:')
  "/>

</xsl:function>

<!--
  Whether an XML node is an ancestor of another node

  @author  Priscilla Walmsley, Datypic
```

```
@version 1.0
@see      http://www.xsltfunctions.com/xsl/functx_is-ancestor.html
@param    $node1 the first node
@param    $node2 the second node
-|->
<xsl:function name="functx:is-ancestor" as="xs:boolean"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="node1" as="node()" />
  <xsl:param name="node2" as="node()" />

  <xsl:sequence select="
    exists($node1 intersect $node2/ancestor::node())
  " />

</xsl:function>

<!--\--
  Whether an XML node is a descendant of another node

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_is-descendant.html
@param    $node1 the first node
@param    $node2 the second node
-|->
<xsl:function name="functx:is-descendant" as="xs:boolean"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="node1" as="node()" />
  <xsl:param name="node2" as="node()" />

  <xsl:sequence select="
    boolean($node2 intersect $node1/ancestor::node())
  " />

</xsl:function>

<!--\--
  Whether a date falls in a leap year

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_is-leap-year.html
@param    $date the date or year
-|->
<xsl:function name="functx:is-leap-year" as="xs:boolean"
           xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    for $year in xs:integer(substring(string($date),1,4))
    return ($year mod 4 = 0 and
            $year mod 100 != 0) or
```



```

        $year mod 400 = 0
    "/>
</xsl:function>

<!--
    Whether an XML node is among the descendants of a sequence, based on contents and attributes

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_is-node-among-descendants-deep-equal.html
    @param    $node the node to test
    @param    $seq the sequence of nodes to search
-->
<xsl:function name="functx:is-node-among-descendants-deep-equal" as="xs:boolean"
               xmlns:functx="http://www.functx.com" >
    <xsl:param name="node" as="node()?" />
    <xsl:param name="seq" as="node(*)" />

    <xsl:sequence select="
        some $nodeInSeq in $seq/descendant-or-self::*/(./*)
        satisfies deep-equal($nodeInSeq,$node)
    "/>
</xsl:function>

<!--
    Whether an XML node is among the descendants of a sequence, based on node identity

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_is-node-among-descendants.html
    @param    $node the node to test
    @param    $seq the sequence of nodes to search
-->
<xsl:function name="functx:is-node-among-descendants" as="xs:boolean"
               xmlns:functx="http://www.functx.com" >
    <xsl:param name="node" as="node()?" />
    <xsl:param name="seq" as="node(*)" />

    <xsl:sequence select="
        some $nodeInSeq in $seq/descendant-or-self::*/(./*)
        satisfies $nodeInSeq is $node
    "/>
</xsl:function>

-->
<!--
    Whether an XML node is in a sequence, based on contents and attributes

    @author   Priscilla Walmsley, Datypic
```

```
@version 1.0
@see      http://www.xsltfunctions.com/xsl/functx_is-node-in-sequence-deep-equal.html
@param    $node the node to test
@param    $seq the sequence of nodes to search
-->
<xsl:function name="functx:is-node-in-sequence-deep-equal" as="xs:boolean">
    <xsl:param name="node" as="node()?" />
    <xsl:param name="seq" as="node(*)" />

    <xsl:sequence select="
        some $nodeInSeq in $seq satisfies deep-equal($nodeInSeq,$node)
    " />

</xsl:function>
<!--

<!--\-
    Whether an XML node is in a sequence, based on node identity

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_is-node-in-sequence.html
@param    $node the node to test
@param    $seq the sequence of nodes to search
-\->
<xsl:function name="functx:is-node-in-sequence" as="xs:boolean"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="node" as="node()?" />
    <xsl:param name="seq" as="node(*)" />

    <xsl:sequence select="
        some $nodeInSeq in $seq satisfies $nodeInSeq is $node
    " />

</xsl:function>

<!--\-
    Whether an atomic value appears in a sequence

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_is-value-in-sequence.html
@param    $value the atomic value to test
@param    $seq the sequence of values to search
-\->
<xsl:function name="functx:is-value-in-sequence" as="xs:boolean"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="value" as="xs:anyAtomicType?" />
    <xsl:param name="seq" as="xs:anyAtomicType*" />

    <xsl:sequence select="
        $value = $seq
    " />

</xsl:function>
```

```
<!--\n
  The last day of the month of a date

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_last-day-of-month.html
@param   $date the date
--\->
<xsl:function name="functx:last-day-of-month" as="xs:date?"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    functx:date(year-from-date(xs:date($date)),
                month-from-date(xs:date($date)),
                functx:days-in-month($date))
  "/>
</xsl:function>

<!--\n
  The last day of the month of a date

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_last-day-of-year.html
@param   $date the date
--\->
<xsl:function name="functx:last-day-of-year" as="xs:date?"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    functx:date(year-from-date(xs:date($date)), 12, 31)
  "/>
</xsl:function>

<!--\n
  The XML node in a sequence that is last in document order

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_last-node.html
@param   $nodes the sequence of nodes
--\->
<xsl:function name="functx:last-node" as="node()?"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node()*" />
```

```
<xsl:sequence select="
  ($nodes/.)[last()]
"/>

</xsl:function>

<!--
  All XML elements that don't have any child elements

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_leaf-elements.html
  @param   $root the root
-->
<xsl:function name="functx:leaf-elements" as="element(*)"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="root" as="node()?" />

  <xsl:sequence select="
    $root/descendant-or-self::*[not(*)]
  "/>

</xsl:function>

<!--
  Trims leading whitespace

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_left-trim.html
  @param   $arg the string to trim
-->
<xsl:function name="functx:left-trim" as="xs:string"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    replace($arg, '^\\s+', '')
  "/>

</xsl:function>

<!--
  The number of lines

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_line-count.html
  @param   $arg the string to test
-->
<xsl:function name="functx:line-count" as="xs:integer"
```

```

        xmlns:functx="http://www.functx.com" >
<xsl:param name="arg" as="xs:string?"/>

<xsl:sequence select="
    count(functx:lines($arg))
"/>

</xsl:function>

<!--
    Split a string into separate lines

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_lines.html
    @param    $arg the string to split
-->
<xsl:function name="functx:lines" as="xs:string*"
        xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?"/>

    <xsl:sequence select="
        tokenize($arg, '(\r\n?|\n\r?)')
    "/>

</xsl:function>

<!--
    The maximum depth of elements in an XML tree

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_max-depth.html
    @param    $root the root to start from
-->
<xsl:function name="functx:max-depth" as="xs:integer?"
        xmlns:functx="http://www.functx.com" >
    <xsl:param name="root" as="node()?"/>

    <xsl:sequence select="
        if ($root/*)
        then max($root/*/functx:max-depth(.)) + 1
        else 1
    "/>

</xsl:function>

<!--
    The maximum value in a sequence, figuring out its type (numeric or string)

    @author   Priscilla Walmsley, Datypic
```

```
@version 1.0
@see      http://www.xsltfunctions.com/xsl/functx_max-determine-type.html
@param    $seq the sequence of values to test
-|->
<xsl:function name="functx:max-determine-type" as="xs:anyAtomicType?"
            xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    if (every $value in $seq satisfies ($value castable as xs:double))
    then max(for $value in $seq return xs:double($value))
    else max(for $value in $seq return xs:string($value))
  " />

</xsl:function>

<!--\--
  The maximum line length

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_max-line-length.html
@param    $arg the string to test
-|->
<xsl:function name="functx:max-line-length" as="xs:integer"
            xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    max(
      for $line in functx:lines($arg)
      return string-length($line)
    )
  " />

</xsl:function>

<!--\--
  The XML node whose typed value is the maximum

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_max-node.html
@param    $nodes the sequence of nodes to test
-|->
<xsl:function name="functx:max-node" as="node()*"
            xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node()*" />

  <xsl:sequence select="
    $nodes[. = max($nodes)]
  " />
```

```
</xsl:function>

<!--
  The maximum of a sequence of values, treating them like strings

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_max-string.html
  @param   $strings the sequence of values
-->
<xsl:function name="functx:max-string" as="xs:string?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="strings" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    max(for $string in $strings return string($string))
  " />
</xsl:function>

<!--
  The minimum value in a sequence, figuring out its type (numeric or string)

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_min-determine-type.html
  @param   $seq the sequence of values to test
-->
<xsl:function name="functx:min-determine-type" as="xs:anyAtomicType?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    if (every $value in $seq satisfies ($value castable as xs:double))
    then min(for $value in $seq return xs:double($value))
    else min(for $value in $seq return xs:string($value))
  " />
</xsl:function>

<!--
  The XML node whose typed value is the minimum

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_min-node.html
  @param   $nodes the sequence of nodes to test
-->
<xsl:function name="functx:min-node" as="node(*)"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node(*)*" />
```

```
<xsl:sequence select="
  $nodes[. = min($nodes)]
"/>

</xsl:function>

<!--
  The minimum of a sequence of strings, ignoring "empty" values

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_min-non-empty-string.html
  @param   $strings the sequence of strings to search
-->
<xsl:function name="functx:min-non-empty-string" as="xs:string?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="strings" as="xs:string*" />

  <xsl:sequence select="
    min($strings[. != ''])
  "/>

</xsl:function>

<!--
  The minimum of a sequence of values, treating them like strings

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_min-string.html
  @param   $strings the sequence of strings
-->
<xsl:function name="functx:min-string" as="xs:string?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="strings" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    min(for $string in $strings return string($string))
  "/>

</xsl:function>

<!--
  Converts a string with format MMDDYYYY (with any delimiters) to a date

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_mmddyyyy-to-date.html
  @param   $dateString the MMDDYYYY string
-->
```



```
<xsl:function name="functx:mmdyyy-to-date" as="xs:date?"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="dateString" as="xs:string?" />

  <xsl:sequence select="
    if (empty($dateString))
    then ()
    else if (not(matches($dateString,
                        '^D*(\d{2})\D*(\d{2})\D*(\d{4})\D*$'))
    then error(xs:QName('functx:Invalid_Date_Format'))
    else xs:date(replace($dateString,
                        '^D*(\d{2})\D*(\d{2})\D*(\d{4})\D*$',
                        '$3-$1-$2'))
  "/>

</xsl:function>

<!--
  The month of a date as an abbreviated word (Jan, Feb, etc.)

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_month-abbrev-en.html
  @param   $date the date
-->
<xsl:function name="functx:month-abbrev-en" as="xs:string?"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
     'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')
    [month-from-date(xs:date($date))]" />

</xsl:function>

<!--
  The month of a date as a word (January, February, etc.)

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_month-name-en.html
  @param   $date the date
-->
<xsl:function name="functx:month-name-en" as="xs:string?"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    ('January', 'February', 'March', 'April', 'May', 'June',
     'July', 'August', 'September', 'October', 'November', 'December')
  "
```

```

    [month-from-date(xs:date($date))]]
"/>

</xsl:function>

<!--\-
    Whether a name matches a list of names or name wildcards

    @author  Priscilla Walmsley, Datypic
    @version 1.0
    @see     http://www.xsltfunctions.com/xsl/functx_name-test.html
    @param   $testname the name to test
    @param   $names the list of names or name wildcards
--\->
<xsl:function name="functx:name-test" as="xs:boolean"
               xmlns:functx="http://www.functx.com" >
    <xsl:param name="testname" as="xs:string?" />
    <xsl:param name="names" as="xs:string*" />

    <xsl:sequence select="
$testname = $names
or
$names = '*'
or
functx:substring-after-if-contains($testname,':') =
    (for $name in $names
    return substring-after($name,'*:') )
or
substring-before($testname,':') =
    (for $name in $names[contains(.,':*')]
    return substring-before($name,'*'))
    "/>

</xsl:function>

<!--\-
    A list of namespaces used in element/attribute names in an XML fragment

    @author  Priscilla Walmsley, Datypic
    @version 1.0
    @see     http://www.xsltfunctions.com/xsl/functx_namespaces-in-use.html
    @param   $root the root node to start from
--\->
<xsl:function name="functx:namespaces-in-use" as="xs:anyURI*"
               xmlns:functx="http://www.functx.com" >
    <xsl:param name="root" as="node()?" />

    <xsl:sequence select="
    distinct-values(
        $root/descendant-or-self::*/(./@*)/namespace-uri(.))
    "/>
```

```
</xsl:function>

<!--
  The next day

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/funcctx_next-day.html
  @param   $date the date
-->
<xsl:function name="funcctx:next-day" as="xs:date?"
              xmlns:funcctx="http://www.funcctx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />

  <xsl:sequence select="
    xs:date($date) + xs:dayTimeDuration('P1D')
  " />
</xsl:function>

<!--
  The XML node kind (element, attribute, text, etc.)

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/funcctx_node-kind.html
  @param   $nodes the node(s) whose kind you want to determine
-->
<xsl:function name="funcctx:node-kind" as="xs:string*"
              xmlns:funcctx="http://www.funcctx.com" >
  <xsl:param name="nodes" as="node(*)*" />

  <xsl:sequence select="
for $node in $nodes
return
if ($node instance of element()) then 'element'
else if ($node instance of attribute()) then 'attribute'
else if ($node instance of text()) then 'text'
else if ($node instance of document-node()) then 'document-node'
else if ($node instance of comment()) then 'comment'
else if ($node instance of processing-instruction())
    then 'processing-instruction'
else 'unknown'
  " />
</xsl:function>

<!--
  Returns any values that appear more than once in a sequence

  @author  Priscilla Walmsley, Datypic
```

```
@version 1.0
@see      http://www.xsltfunctions.com/xsl/functx_non-distinct-values.html
@param    $seq the sequence of values
-|->
<xsl:function name="functx:non-distinct-values" as="xs:anyAtomicType*"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    for $val in distinct-values($seq)
    return $val[count($seq[. = $val]) > 1]
  " />

</xsl:function>

<!--\--
  The number of regions that match a pattern

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_number-of-matches.html
@param    $arg the string to test
@param    $pattern the regular expression
-|->
<xsl:function name="functx:number-of-matches" as="xs:integer"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="pattern" as="xs:string" />

  <xsl:sequence select="
    count(tokenize($arg,$pattern)) - 1
  " />

</xsl:function>

<!--\--
  Resolves a relative URI and references it, returning an XML document

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_open-ref-document.html
@param    $refNode a node whose value is a relative URI reference
-|->
<xsl:function name="functx:open-ref-document" as="document-node()"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="refNode" as="node()" />

  <xsl:sequence select="
    if (base-uri($refNode))
    then doc(resolve-uri($refNode, base-uri($refNode)))
    else doc(resolve-uri($refNode))
  " />
```

</xsl:function>

<!--
Reformats a number as an ordinal number, e.g. 1st, 2nd, 3rd.

@author Priscilla Walmsley, Datypic
@version 1.0
@see http://www.xsltfunctions.com/xsl/functx_ordinal-number-en.html
@param \$num the number

-->
<xsl:function name="functx:ordinal-number-en" as="xs:string"
xmlns:functx="http://www.functx.com" >
 <xsl:param name="num" as="xs:integer?" />

 <xsl:sequence select="
 concat(xs:string(\$num),
 if (matches(xs:string(\$num),'[04-9]\$|1[1-3]\$')) then 'th'
 else if (ends-with(xs:string(\$num),'1')) then 'st'
 else if (ends-with(xs:string(\$num),'2')) then 'nd'
 else if (ends-with(xs:string(\$num),'3')) then 'rd'
 else '')
 "/>

</xsl:function>

<!--
Pads an integer to a desired length by adding leading zeros

@author Priscilla Walmsley, Datypic
@version 1.0
@see http://www.xsltfunctions.com/xsl/functx_pad-integer-to-length.html
@param \$integerToPad the integer to pad
@param \$length the desired length

-->
<xsl:function name="functx:pad-integer-to-length" as="xs:string"
xmlns:functx="http://www.functx.com" >
 <xsl:param name="integerToPad" as="xs:anyAtomicType?" />
 <xsl:param name="length" as="xs:integer" />

 <xsl:sequence select="
 if (\$length < string-length(string(\$integerToPad)))
 then error(xs:QName('functx:Integer_Longer_Than_Length'))
 else concat
 (functx:repeat-string(
 '0', \$length - string-length(string(\$integerToPad))),
 string(\$integerToPad))
 "/>

</xsl:function>

```
<!--\-
  Pads a string to a desired length

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_pad-string-to-length.html
@param   $stringToPad the string to pad
@param   $padChar the character(s) to use as padding
@param   $length the desired length
--\->
<xsl:function name="functx:pad-string-to-length" as="xs:string"
      xmlns:functx="http://www.functx.com" >
  <xsl:param name="stringToPad" as="xs:string?" />
  <xsl:param name="padChar" as="xs:string" />
  <xsl:param name="length" as="xs:integer" />

  <xsl:sequence select="
    substring(
      string-join (
        ($stringToPad, for $i in (1 to $length) return $padChar)
        ,'' )
      ,1,$length)
  "/>

</xsl:function>

<!--\-
  A unique path to an XML node (or sequence of nodes)

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_path-to-node-with-pos.html
@param   $node the node sequence
--\->
<xsl:function name="functx:path-to-node-with-pos" as="xs:string"
      xmlns:functx="http://www.functx.com" >
  <xsl:param name="node" as="node()?" />

  <xsl:variable name="names" as="xs:string*">
    <xsl:for-each select="$node/ancestor-or-self::*">
      <xsl:variable name="ancestor" select="."/>
      <xsl:variable name="sibsOfSameName"
        select="$ancestor/../*[name() = name($ancestor)]"/>
      <xsl:sequence select="concat(name($ancestor),
        if (count($sibsOfSameName) <= 1)
        then ''
        else concat(
          '[' ,functx:index-of-node($sibsOfSameName,$ancestor),']') )"/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:sequence select="string-join($names, '/')"/>

</xsl:function>
```

```
<!--\-
  A path to an XML node (or sequence of nodes)

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_path-to-node.html
@param   $nodes the node sequence
--\->
<xsl:function name="functx:path-to-node" as="xs:string*"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node(*)*" />

  <xsl:sequence select="
$nodes/string-join(ancestor-or-self::* /name(.), '/')
" />

</xsl:function>

<!--\-
  Whether an XML node precedes another without being its ancestor

@author  W3C XML Query Working Group
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_precedes-not-ancestor.html
@param   $a the first node
@param   $b the second node
--\->
<xsl:function name="functx:precedes-not-ancestor" as="xs:boolean"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="a" as="node(*)*" />
  <xsl:param name="b" as="node(*)*" />

  <xsl:sequence select="
  $a &lt;&lt; $b and empty($a intersect $b/ancestor::node())
" />

</xsl:function>

<!--\-
  The previous day

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_previous-day.html
@param   $date the date
--\->
<xsl:function name="functx:previous-day" as="xs:date?"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="date" as="xs:anyAtomicType?" />
```

```
<xsl:sequence select="
  xs:date($date) - xs:dayTimeDuration('P1D')
"/>

</xsl:function>

<!--
  Removes attributes from an XML fragment, based on name

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_remove-attributes-deep.html
  @param   $nodes the root(s) to start from
  @param   $names the names of the attributes to remove, or * for all attributes
-->
<xsl:function name="functx:remove-attributes-deep" as="node(*)"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node(*)"/>
  <xsl:param name="names" as="xs:string*" />

  <xsl:for-each select="$nodes">
    <xsl:choose>
      <xsl:when test=". instance of element()">
        <xsl:element name="{node-name(.)}">
          <xsl:sequence select="
            (@*[not(functx:name-test(name(),$names)]),
            functx:remove-attributes-deep(node(), $names))"/>
        </xsl:element>
      </xsl:when>
      <xsl:when test=". instance of document-node()">
        <xsl:document>
          <xsl:sequence select="
            functx:remove-attributes-deep(node(), $names)" />
        </xsl:document>
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:function>

<!--
  Removes attributes from an XML element, based on name

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_remove-attributes.html
  @param   $element the element(s) from which to remove the attributes
  @param   $names the names of the attributes to remove, or * for all attributes
-->
```



```
<xsl:function name="functx:remove-attributes" as="element()"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="elements" as="element(*)" />
  <xsl:param name="names" as="xs:string*" />

  <xsl:for-each select="$elements">
    <xsl:element name="{node-name(.)}">
      <xsl:sequence
        select="(@*[not(functx:name-test(name(),$names)]),
          node())"/>
    </xsl:element>
  </xsl:for-each>
</xsl:function>

<!--
  Removes descendant elements from an XML node, based on name

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_remove-elements-deep.html
  @param   $nodes root(s) to start from
  @param   $names the names of the elements to remove
-->
<xsl:function name="functx:remove-elements-deep" as="node(*)"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node(*)" />
  <xsl:param name="names" as="xs:string*" />

  <xsl:for-each select="$nodes">
    <xsl:choose>
      <xsl:when test=". instance of element()">
        <xsl:if test="not(functx:name-test(name(),$names))">
          <xsl:element name="{node-name(.)}">
            <xsl:sequence select="@*,
              functx:remove-elements-deep(node(), $names)"/>
          </xsl:element>
        </xsl:if>
      </xsl:when>
      <xsl:when test=". instance of document-node()">
        <xsl:document>
          <xsl:sequence select="
            functx:remove-elements-deep(node(), $names)"/>
        </xsl:document>
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:function>
```

```
<!--\-
  Removes descendant XML elements but keeps their content

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_remove-elements-not-contents.html
@param   $nodes the root(s) to start from
@param   $names the names of the elements to remove
--\->
<xsl:function name="functx:remove-elements-not-contents" as="node(*)"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="nodes" as="node(*)" />
  <xsl:param name="names" as="xs:string*" />

  <xsl:for-each select="$nodes">
    <xsl:choose>
      <xsl:when test=". instance of element()">
        <xsl:choose>
          <xsl:when test="functx:name-test(name(),$names)">
            <xsl:sequence select="
              functx:remove-elements-not-contents(node(), $names)"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:element name="{node-name(.)}">
              <xsl:sequence select="@*,
                functx:remove-elements-not-contents(node(),$names)"/>
            </xsl:element>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:when test=". instance of document-node()">
        <xsl:document>
          <xsl:sequence select="
            functx:remove-elements-not-contents(node(), $names)"/>
        </xsl:document>
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:function>

<!--\-
  Removes child elements from an XML node, based on name

@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_remove-elements.html
@param   $element the element(s) from which you wish to remove the children
@param   $names the names of the child elements to remove
```

```
-\->
<xsl:function name="functx:remove-elements" as="element(*)"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="elements" as="element(*)"/>
  <xsl:param name="names" as="xs:string*" />

  <xsl:for-each select="$elements">
    <xsl:element name="{node-name(.)}">
      <xsl:sequence select="(@*,
        node()[not(functx:name-test(name(),$names)])]" />
    </xsl:element>
  </xsl:for-each>
</xsl:function>

<!--\-
  Repeats a string a given number of times

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_repeat-string.html
  @param   $stringToRepeat the string to repeat
  @param   $count the desired number of copies
--\->
<xsl:function name="functx:repeat-string" as="xs:string"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="stringToRepeat" as="xs:string?" />
  <xsl:param name="count" as="xs:integer" />

  <xsl:sequence select="
    string-join((for $i in 1 to $count return $stringToRepeat),
      '')
  " />
</xsl:function>

<!--\-
  Replaces the beginning of a string, up to a matched pattern

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_replace-beginning.html
  @param   $arg the entire string to change
  @param   $pattern the pattern of characters to replace up to
  @param   $replacement the replacement string
--\->
<xsl:function name="functx:replace-beginning" as="xs:string"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="pattern" as="xs:string" />
  <xsl:param name="replacement" as="xs:string" />
```

```
<xsl:sequence select="
  replace($arg, concat('^.*?', $pattern), $replacement)
"/>

</xsl:function>

<!--
  Updates the content of one or more elements

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_replace-element-values.html
  @param   $elements the elements whose content you wish to replace
  @param   $values the replacement values
-->
<xsl:function name="functx:replace-element-values" as="element(*)"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="elements" as="element(*)"/>
  <xsl:param name="values" as="xs:anyAtomicType*" />

  <xsl:for-each select="$elements">
    <xsl:variable name="seq" select="position()" />
    <xsl:element name="{node-name(.)}">
      <xsl:sequence select="@*, $values[$seq]" />
    </xsl:element>
  </xsl:for-each>
</xsl:function>

<!--
  Replaces the first match of a pattern

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_replace-first.html
  @param   $arg the entire string to change
  @param   $pattern the pattern of characters to replace
  @param   $replacement the replacement string
-->
<xsl:function name="functx:replace-first" as="xs:string"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="pattern" as="xs:string" />
  <xsl:param name="replacement" as="xs:string" />

  <xsl:sequence select="
    replace($arg, concat('(^.*?)', $pattern),
      concat('$1', $replacement))
  "/>

</xsl:function>
```

```
<!--  
    Performs multiple replacements, using pairs of replace parameters  
  
    @author   Priscilla Walmsley, Datypic  
    @version  1.0  
    @see      http://www.xsltfunctions.com/xsl/functx_replace-multi.html  
    @param    $arg the string to manipulate  
    @param    $changeFrom the sequence of strings or patterns to change from  
    @param    $changeTo the sequence of strings to change to  
-->  
<xsl:function name="functx:replace-multi" as="xs:string?"  
    xmlns:functx="http://www.functx.com" >  
    <xsl:param name="arg" as="xs:string?"/>  
    <xsl:param name="changeFrom" as="xs:string*"/>  
    <xsl:param name="changeTo" as="xs:string*"/>  
  
    <xsl:sequence select="  
        if (count($changeFrom) > 0)  
        then functx:replace-multi(  
            replace($arg, $changeFrom[1],  
                functx:if-absent($changeTo[1], '')),  
            $changeFrom[position() > 1],  
            $changeTo[position() > 1])  
        else $arg  
    "/>  
</xsl:function>  
  
<!--  
    Reverses the order of characters  
  
    @author   Priscilla Walmsley, Datypic  
    @version  1.0  
    @see      http://www.xsltfunctions.com/xsl/functx_reverse-string.html  
    @param    $arg the string to reverse  
-->  
<xsl:function name="functx:reverse-string" as="xs:string"  
    xmlns:functx="http://www.functx.com" >  
    <xsl:param name="arg" as="xs:string?"/>  
  
    <xsl:sequence select="  
        codepoints-to-string(reverse(string-to-codepoints($arg)))  
    "/>  
</xsl:function>  
  
<!--  
    Trims trailing whitespace  
  
    @author   Priscilla Walmsley, Datypic  
    @version  1.0
```

```
@see      http://www.xsltfunctions.com/xsl/functx_right-trim.html
@param    $arg the string to trim
-|->
<xsl:function name="functx:right-trim" as="xs:string"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    replace($arg,'\\s+$','')
  "/>
</xsl:function>

<!--
  Returns the scheme from a URI

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see      http://www.xsltfunctions.com/xsl/functx_scheme-from-uri.html
  @param    $uri the URI
  -|->
<xsl:function name="functx:scheme-from-uri" as="xs:string?"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="uri" as="xs:string?" />

  <xsl:sequence select="
    substring-before($uri,':')
  "/>
</xsl:function>

<!--
  Whether two sequences have the same XML node content and/or values

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see      http://www.xsltfunctions.com/xsl/functx_sequence-deep-equal.html
  @param    $seq1 the first sequence
  @param    $seq2 the second sequence
  -|->
<xsl:function name="functx:sequence-deep-equal" as="xs:boolean"
          xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq1" as="item()*" />
  <xsl:param name="seq2" as="item()*" />

  <xsl:sequence select="
    every $i in 1 to max((count($seq1),count($seq2)))
    satisfies deep-equal($seq1[$i],$seq2[$i])
  "/>
</xsl:function>
```

```
<!--
  Whether two sequences contain the same XML nodes, regardless of order

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_sequence-node-equal-any-order.html
  @param   $seq1 the first sequence of nodes
  @param   $seq2 the second sequence of nodes
-->
<xsl:function name="functx:sequence-node-equal-any-order" as="xs:boolean"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq1" as="node(*)"/>
  <xsl:param name="seq2" as="node(*)"/>

  <xsl:sequence select="
    not( ($seq1 except $seq2, $seq2 except $seq1))
  "/>

</xsl:function>

<!--
  Whether two sequences contain the same XML nodes, in the same order

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_sequence-node-equal.html
  @param   $seq1 the first sequence of nodes
  @param   $seq2 the second sequence of nodes
-->
<xsl:function name="functx:sequence-node-equal" as="xs:boolean"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq1" as="node(*)"/>
  <xsl:param name="seq2" as="node(*)"/>

  <xsl:sequence select="
    every $i in 1 to max((count($seq1),count($seq2)))
    satisfies $seq1[$i] is $seq2[$i]
  "/>

</xsl:function>

<!--
  The sequence type that represents a sequence of nodes or values

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_sequence-type.html
  @param   $items the items whose sequence type you want to determine
-->
<xsl:function name="functx:sequence-type" as="xs:string"
              xmlns:functx="http://www.functx.com" >
```

```
<xsl:param name="items" as="item()*"/>

<xsl:sequence select="
concat(
  if (empty($items))
  then 'empty-sequence()'
  else if (every $val in $items
    satisfies $val instance of xs:anyAtomicType)
  then if (count(distinct-values(funcctx:atomic-type($items)))
    > 1)
  then 'xs:anyAtomicType'
  else funcctx:atomic-type($items[1])
  else if (some $val in $items
    satisfies $val instance of xs:anyAtomicType)
  then 'item()'
  else if (count(distinct-values(funcctx:node-kind($items))) > 1)
  then 'node()'
  else concat(funcctx:node-kind($items[1]),'()')
  ,
  if (count($items) > 1)
  then '+' else ''
)
"/>

</xsl:function>

<!--
  The siblings of an XML element that have the same name

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/funcctx_siblings-same-name.html
  @param   $element the node
-->
<xsl:function name="funcctx:siblings-same-name" as="element()*"
  xmlns:funcctx="http://www.funcctx.com" >
  <xsl:param name="element" as="element()?" />

  <xsl:sequence select="
    $element/../*[node-name(.) = node-name($element)]
    except $element
  "/>

</xsl:function>

<!--
  The siblings of an XML node

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/funcctx_siblings.html
  @param   $node the node
-->
```



```
<xsl:function name="functx:siblings" as="node()*"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="node" as="node()?" />

  <xsl:sequence select="
    $node/../../node() except $node
  " />
</xsl:function>

<!--\--
  Sorts a sequence of numeric values or nodes

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_sort-as-numeric.html
  @param   $seq the sequence to sort
--\-->
<xsl:function name="functx:sort-as-numeric" as="item()*"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq" as="item()*" />

  <xsl:for-each select="$seq">
    <xsl:sort select="number(.)" />
    <xsl:copy-of select="." />
  </xsl:for-each>
</xsl:function>

<!--\--
  Sorts a sequence of values or nodes regardless of capitalization

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_sort-case-insensitive.html
  @param   $seq the sequence to sort
--\-->
<xsl:function name="functx:sort-case-insensitive" as="item()*"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq" as="item()*" />

  <xsl:for-each select="$seq">
    <xsl:sort select="upper-case(string(.))" />
    <xsl:copy-of select="." />
  </xsl:for-each>
</xsl:function>

<!--\--
  Sorts a sequence of nodes in document order
```

```
@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_sort-document-order.html
@param   $seq the sequence to sort
-|->
<xsl:function name="functx:sort-document-order" as="node(*)"
            xmlns:functx="http://www.functx.com" >
  <xsl:param name="seq" as="node(*)"/>

  <xsl:sequence select="
    $seq/.
  "/>

</xsl:function>

<!--
  Sorts a sequence of values or nodes

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_sort.html
  @param   $seq the sequence to sort
  -|->
  <xsl:function name="functx:sort" as="item(*)"
            xmlns:functx="http://www.functx.com" >
    <xsl:param name="seq" as="item(*)"/>

    <xsl:for-each select="$seq">
      <xsl:sort select="."/>
      <xsl:copy-of select="."/>
    </xsl:for-each>

  </xsl:function>

  <!--
    Performs substring-after, returning the entire string if it does not contain the delimiter

    @author  Priscilla Walmsley, Datypic
    @version 1.0
    @see     http://www.xsltfunctions.com/xsl/functx_substring-after-if-contains.html
    @param   $arg the string to substring
    @param   $delim the delimiter
    -|->
    <xsl:function name="functx:substring-after-if-contains" as="xs:string?"
            xmlns:functx="http://www.functx.com" >
      <xsl:param name="arg" as="xs:string?"/>
      <xsl:param name="delim" as="xs:string"/>

      <xsl:sequence select="
        if (contains($arg,$delim))
        then substring-after($arg,$delim)
        else $arg
      ">
```

```
"/>

</xsl:function>

<!--\-
  The substring after the last text that matches a regex

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_substring-after-last-match.html
  @param   $arg the string to substring
  @param   $regex the regular expression
--\->
<xsl:function name="functx:substring-after-last-match" as="xs:string"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="regex" as="xs:string" />

  <xsl:sequence select="
    replace($arg,concat('^.*',$regex),'')
  "/>

</xsl:function>

<!--\-
  The substring after the last occurrence of a delimiter

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_substring-after-last.html
  @param   $arg the string to substring
  @param   $delim the delimiter
--\->
<xsl:function name="functx:substring-after-last" as="xs:string"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="delim" as="xs:string" />

  <xsl:sequence select="
    replace ($arg,concat('^.*',functx:escape-for-regex($delim)),'')
  "/>

</xsl:function>

<!--\-
  The substring after the first text that matches a regex

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_substring-after-match.html
  @param   $arg the string to substring
```

```
@param  $regex the regular expression
-|->
<xsl:function name="functx:substring-after-match" as="xs:string?"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?"/>
    <xsl:param name="regex" as="xs:string"/>

    <xsl:sequence select="
        replace($arg,concat('^.*?', $regex), '')
    "/>
</xsl:function>

<!--
    Performs substring-before, returning the entire string if it does not contain the delimiter

    @author  Priscilla Walmsley, Datypic
    @version 1.0
    @see      http://www.xsltfunctions.com/xsl/functx_substring-before-if-contains.html
    @param    $arg the string to substring
    @param    $delim the delimiter
-|->
<xsl:function name="functx:substring-before-if-contains" as="xs:string?"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?"/>
    <xsl:param name="delim" as="xs:string"/>

    <xsl:sequence select="
        if (contains($arg,$delim))
        then substring-before($arg,$delim)
        else $arg
    "/>
</xsl:function>

<!--
    The substring after the first text that matches a regex

    @author  Priscilla Walmsley, Datypic
    @version 1.0
    @see      http://www.xsltfunctions.com/xsl/functx_substring-before-last-match.html
    @param    $arg the string to substring
    @param    $regex the regular expression
-|->
<xsl:function name="functx:substring-before-last-match" as="xs:string?"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="arg" as="xs:string?"/>
    <xsl:param name="regex" as="xs:string"/>

    <xsl:sequence select="
        replace($arg,concat('^(.*)', $regex, '.*'), '$1')
    "/>
```

```
</xsl:function>

<!--
  The substring before the last occurrence of a delimiter

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_substring-before-last.html
  @param   $arg the string to substring
  @param   $delim the delimiter
-->
<xsl:function name="functx:substring-before-last" as="xs:string"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="delim" as="xs:string" />

  <xsl:sequence select="
    if (matches($arg, functx:escape-for-regex($delim)))
    then replace($arg,
      concat('^(.*)', functx:escape-for-regex($delim), '.*'),
      '$1')
    else ''
  " />
</xsl:function>

<!--
  The substring before the last text that matches a regex

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_substring-before-match.html
  @param   $arg the string to substring
  @param   $regex the regular expression
-->
<xsl:function name="functx:substring-before-match" as="xs:string"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />
  <xsl:param name="regex" as="xs:string" />

  <xsl:sequence select="
    tokenize($arg,$regex)[1]
  " />
</xsl:function>

<!--
  Construct a time from an hour, minute and second

  @author  Priscilla Walmsley, Datypic
```

```
@version 1.0
@see      http://www.xsltfunctions.com/xsl/functx_time.html
@param    $hour the hour
@param    $minute the minute
@param    $second the second
-|->
<xsl:function name="functx:time" as="xs:time"
      xmlns:functx="http://www.functx.com" >
  <xsl:param name="hour" as="xs:anyAtomicType"/>
  <xsl:param name="minute" as="xs:anyAtomicType"/>
  <xsl:param name="second" as="xs:anyAtomicType"/>

  <xsl:sequence select="
    xs:time(
      concat(
        functx:pad-integer-to-length(xs:integer($hour),2),':',
        functx:pad-integer-to-length(xs:integer($minute),2),':',
        functx:pad-integer-to-length(xs:integer($second),2)))
  "/>

</xsl:function>

<!--
  Converts an xs:dayTimeDuration into a timezone like "-05:00" or "Z"

@author   Priscilla Walmsley, Datypic
@version  1.0
@see      http://www.xsltfunctions.com/xsl/functx_timezone-from-duration.html
@param    $duration the duration
-|->
<xsl:function name="functx:timezone-from-duration" as="xs:string"
      xmlns:functx="http://www.functx.com" >
  <xsl:param name="duration" as="xs:dayTimeDuration"/>

  <xsl:sequence select="
    if (string($duration) = ('PT0S','-PT0S'))
    then 'Z'
    else if (matches(string($duration),'-PT[1-9]H'))
    then replace(string($duration),'PT([1-9])H','0$1:00')
    else if (matches(string($duration),'PT[1-9]H'))
    then replace(string($duration),'PT([1-9])H','+$1:00')
    else if (matches(string($duration),'-PT1[0-4]H'))
    then replace(string($duration),'PT(1[0-4])H','$1:00')
    else if (matches(string($duration),'PT1[0-4]H'))
    then replace(string($duration),'PT(1[0-4])H','+$1:00')
    else error(xs:QName('functx:Invalid_Duration_Value'))
  "/>

</xsl:function>

<!--
  The total number of days in a dayTimeDuration
```

```
@author  Priscilla Walmsley, Datypic
@version 1.0
@see     http://www.xsltfunctions.com/xsl/functx_total-days-from-duration.html
@param   $duration the duration
-|->
<xsl:function name="functx:total-days-from-duration" as="xs:decimal?"
              xmlns:functx="http://www.functx.com" >
  <xsl:param name="duration" as="xs:dayTimeDuration?" />

  <xsl:sequence select="
    $duration div xs:dayTimeDuration('P1D')
  " />

</xsl:function>

<!--\--
  The total number of hours in a dayTimeDuration

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_total-hours-from-duration.html
  @param   $duration the duration
  -|->
  <xsl:function name="functx:total-hours-from-duration" as="xs:decimal?"
                xmlns:functx="http://www.functx.com" >
    <xsl:param name="duration" as="xs:dayTimeDuration?" />

    <xsl:sequence select="
      $duration div xs:dayTimeDuration('PT1H')
    " />

  </xsl:function>

  <!--\--
    The total number of minutes in a dayTimeDuration

    @author  Priscilla Walmsley, Datypic
    @version 1.0
    @see     http://www.xsltfunctions.com/xsl/functx_total-minutes-from-duration.html
    @param   $duration the duration
    -|->
    <xsl:function name="functx:total-minutes-from-duration" as="xs:decimal?"
                  xmlns:functx="http://www.functx.com" >
      <xsl:param name="duration" as="xs:dayTimeDuration?" />

      <xsl:sequence select="
        $duration div xs:dayTimeDuration('PT1M')
      " />

    </xsl:function>
```

```
<!--  
  The total number of months in a yearMonthDuration  
  
  @author  Priscilla Walmsley, Datypic  
  @version 1.0  
  @see     http://www.xsltfunctions.com/xsl/functx_total-months-from-duration.html  
  @param   $duration the duration  
-->  
<xsl:function name="functx:total-months-from-duration" as="xs:decimal?"  
  xmlns:functx="http://www.functx.com" >  
  <xsl:param name="duration" as="xs:yearMonthDuration?" />  
  
  <xsl:sequence select="  
    $duration div xs:yearMonthDuration('P1M')  
  " />  
  
</xsl:function>  
  
<!--  
  The total number of seconds in a dayTimeDuration  
  
  @author  Priscilla Walmsley, Datypic  
  @version 1.0  
  @see     http://www.xsltfunctions.com/xsl/functx_total-seconds-from-duration.html  
  @param   $duration the duration  
-->  
<xsl:function name="functx:total-seconds-from-duration" as="xs:decimal?"  
  xmlns:functx="http://www.functx.com" >  
  <xsl:param name="duration" as="xs:dayTimeDuration?" />  
  
  <xsl:sequence select="  
    $duration div xs:dayTimeDuration('PT1S')  
  " />  
  
</xsl:function>  
  
<!--  
  The total number of years in a yearMonthDuration  
  
  @author  Priscilla Walmsley, Datypic  
  @version 1.0  
  @see     http://www.xsltfunctions.com/xsl/functx_total-years-from-duration.html  
  @param   $duration the duration  
-->  
<xsl:function name="functx:total-years-from-duration" as="xs:decimal?"  
  xmlns:functx="http://www.functx.com" >  
  <xsl:param name="duration" as="xs:yearMonthDuration?" />  
  
  <xsl:sequence select="  
    $duration div xs:yearMonthDuration('P1Y')  
  " />
```



```
</xsl:function>

<!--
  Trims leading and trailing whitespace

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_trim.html
  @param   $arg the string to trim
-->
<xsl:function name="functx:trim" as="xs:string"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    replace(replace($arg,'s+$',''),'^s+', '')
  " />
</xsl:function>

<!--
  Updates the attribute value of an XML element

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_update-attributes.html
  @param   $elements the element(s) for which you wish to update the attribute
  @param   $attrNames the name(s) of the attribute(s) to add
  @param   $attrValues the value(s) of the attribute(s) to add
-->
<xsl:function name="functx:update-attributes" as="element()?"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="elements" as="element(*)" />
  <xsl:param name="attrNames" as="xs:QName*" />
  <xsl:param name="attrValues" as="xs:anyAtomicType*" />

  <xsl:for-each select="$elements">
    <xsl:variable name="element" select="."/>
    <xsl:copy>
      <xsl:for-each select="$attrNames">
        <xsl:variable name="seq" select="position()" />
        <xsl:if test="$element/@*[node-name(.) = current()]">
          <xsl:attribute name="{.}"
            namespace="{namespace-uri-from-QName(.)}"
            select="$attrValues[$seq]" />
        </xsl:if>
      </xsl:for-each>
      <xsl:copy-of select="@*[not(node-name(.) = $attrNames)]|node()" />
    </xsl:copy>
  </xsl:for-each>
</xsl:function>
```

```
</xsl:function>

<!--
  The values in one sequence that aren't in another sequence

  @author  W3C XML Query Working Group
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_value-except.html
  @param   $arg1 the first sequence
  @param   $arg2 the second sequence
-->
<xsl:function name="functx:value-except" as="xs:anyAtomicType*"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg1" as="xs:anyAtomicType*" />
  <xsl:param name="arg2" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    distinct-values($arg1[not(= $arg2)])
  "/>
</xsl:function>

<!--
  The intersection of two sequences of values

  @author  W3C XML Query Working Group
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_value-intersect.html
  @param   $arg1 the first sequence
  @param   $arg2 the second sequence
-->
<xsl:function name="functx:value-intersect" as="xs:anyAtomicType*"
  xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg1" as="xs:anyAtomicType*" />
  <xsl:param name="arg2" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    distinct-values($arg1[= $arg2])
  "/>
</xsl:function>

<!--
  The union of two sequences of values

  @author  W3C XML Query Working Group
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_value-union.html
  @param   $arg1 the first sequence
  @param   $arg2 the second sequence
-->
```

```
<xsl:function name="functx:value-union" as="xs:anyAtomicType*"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg1" as="xs:anyAtomicType*" />
  <xsl:param name="arg2" as="xs:anyAtomicType*" />

  <xsl:sequence select="
    distinct-values(($arg1, $arg2))
  " />
</xsl:function>

<!--
  The number of words

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_word-count.html
  @param   $arg the string to measure
-->
<xsl:function name="functx:word-count" as="xs:integer"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    count(tokenize($arg, '\W+') [. != ''])
  " />
</xsl:function>

<!--
  Turns a string of words into camelCase

  @author  Priscilla Walmsley, Datypic
  @version 1.0
  @see     http://www.xsltfunctions.com/xsl/functx_words-to-camel-case.html
  @param   $arg the string to modify
-->
<xsl:function name="functx:words-to-camel-case" as="xs:string"
    xmlns:functx="http://www.functx.com" >
  <xsl:param name="arg" as="xs:string?" />

  <xsl:sequence select="
    string-join((tokenize($arg, '\s+')[1],
      for $word in tokenize($arg, '\s+')[position() > 1]
      return functx:capitalize-first($word))
    , '')
  " />
</xsl:function>

<!--
```

```

    Wraps a sequence of atomic values in XML elements

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_wrap-values-in-elements.html
    @param    $values the values to wrap in elements
    @param    $elementName the name of the elements to construct
    -\->
<xsl:function name="functx:wrap-values-in-elements" as="element(*)*"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="values" as="xs:anyAtomicType*" />
    <xsl:param name="elementName" as="xs:QName" />

    <xsl:for-each select="$values">
        <xsl:element name="{ $elementName }"
            namespace="{ namespace-uri-from-QName( $elementName ) }">
            <xsl:sequence select="."/>
        </xsl:element>
    </xsl:for-each>

</xsl:function>

<!--\--
    Construct a yearMonthDuration from a number of years and months

    @author   Priscilla Walmsley, Datypic
    @version  1.0
    @see      http://www.xsltfunctions.com/xsl/functx_yearmonthduration.html
    @param    $years the number of years
    @param    $months the number of months
    -\->
<xsl:function name="functx:yearMonthDuration" as="xs:yearMonthDuration"
    xmlns:functx="http://www.functx.com" >
    <xsl:param name="years" as="xs:decimal?" />
    <xsl:param name="months" as="xs:integer?" />

    <xsl:sequence select="
        (xs:yearMonthDuration('P1M') * functx:if-empty($months,0)) +
        (xs:yearMonthDuration('P1Y') * functx:if-empty($years,0))
    " />

</xsl:function>
-->

</xsl:stylesheet>
```

2.2 - ISM-Rollup.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl"
    xmlns:cve="urn:us:gov:ic:cve"
    xmlns:catt="urn:us:gov:ic:taxonomy:catt:tetragraph"
    xmlns:util="urn:us:gov:ic:ism-rollup:xsl:util"
    xmlns:arh="urn:us:gov:ic:arh"
    xmlns:functx="http://www.functx.com"
    xmlns:ism-func="urn:us:gov:ic:ism:functions"
    xmlns:banner="banner.fn"
    xmlns:ntk="urn:us:gov:ic:ntk"
    exclude-result-prefixes="xs xd cve util catt banner ism-func functx"
    version="2.0">

    <xsl:import href="functx-1.0-doc-2007-01.xsl"/>
    <xsl:import href="../../XSL/ISM/IC-ISM-Functions.xsl"/>

    <xd:doc scope="stylesheet">
        <xd:desc>
            <xd:p>
                <xd:b>Created on:</xd:b> Aug 11, 2018</xd:p>
            <xd:p>
                <xd:b>Author:</xd:b> IC-CIO</xd:p>
            <xd:p>Implementation of security markings Rollup using XSLT 2.0. </xd:p>
            <xd:p>Depends on ISMCAT and ISM CVEs and ISMCAT Taxonomy. </xd:p>
            <xd:p>example invocations</xd:p>
            <xd:ul>
                <xd:li>
                    <xd:p>Invocation using dummy paths but showing all parameters </xd:p>
                    <xd:pre>java -jar PathTo/saxon9he.jar -xsl:PathTo/ISM-Rollup.xsl -o:PathTo/OutputRolledUp.xml -s:PathTo/SomeSource.xml \
derivedFrom='Audit Record Portion markings' \
derivativelyClassifiedBy='Bill Smith Govt Accreditor of X' \
pathToISMCATCVE='../../CVE/ISMCAT/' \
pathToISM CVE='../../CVE/ISM/' \
pathToISMCATTaxonomy='../../Taxonomy/ISMCAT/'
</xd:pre>
                </xd:li>

            </xd:ul>
            <xd:p>Utilizes functions from http://www.xsltfunctions.com an GNU Lesser General Public
License as published by the Free Software Foundation; either version 2.1 of the License. The functions are in
A separate file imported to this XSL. None are customized they are used as is.</xd:p>
        </xd:desc>
    </xd:doc>

    <xsl:param name="derivedFrom"
        select="'Portion markings of document sent to rollup Tool'"/>
    <xsl:param name="derivativelyClassifiedBy"
        select="'Human who accredited use of rollup tool in production'"/>
    <xsl:param name="pathToISMCATCVE" select="'../../CVE/ISMCAT/'"/>
    <xsl:param name="pathToISM CVE" select="'../../CVE/ISM/'"/>
    <!-- DY: migrated to IC-ISM-Functions.xsl -->
```

```

<!--xsl:param name="pathToISMCATTaxonomy" select="'../../Taxonomy/ISMCAT/'"></xsl:param-->

<xsl:output method="xml" indent="yes"/>

    <xsl:key name="classification"
        match="*[@ism:classification and util:contributesToRollup(.)]"
        use="@ism:classification"/>
    <xsl:variable name="FGIOpenCVE"
        select="document(concat($pathToISMCATCVE, 'CVEEnumISMCATFGIOpen.xml'))//cve:CVE/cve:Enumeration"/>
    <xsl:variable name="RelCVE"
        select="document(concat($pathToISMCATCVE, 'CVEEnumISMCATRelTo.xml'))//cve:CVE/cve:Enumeration"/>
    <xsl:variable name="disseminationControlsCVE"
        select="document(concat($pathToISM CVE, 'CVEEnumISMDissem.xml'))//cve:CVE/cve:Enumeration"/>

    <!-- lifted from ISM -->
    <xsl:variable name="partTags" select="util:partTags(*)"/>

    <xsl:variable name="countFdrPortions" select="util:countFdrPortions(*)"/>

    <!-- DY: migrated to IC-ISM-Functions.xsl -->
    <!--xsl:variable name="catt"
        select="document(concat($pathToISMCATTaxonomy, 'TetragraphTaxonomyDenormalized.xml'))"/-->

    <!-- DY: migrated to IC-ISM-Functions.xsl -->
    <!--xsl:variable name="cattMappings" select="$catt//catt:Tetragraph"/-->

    <xsl:variable name="tetragraphList"
        select="document(concat($pathToISMCATCVE, 'CVEEnumISMCATTetragraph.xml'))//cve:CVE/cve:Enumeration/cve:Term/cve:Value"/>

    <!-- DY: migrated to IC-ISM-Functions.xsl -->
    <!--xsl:variable name="decomposableTetraElems"
        select="$cattMappings[@decomposable[. = 'Yes' or . = 'NA']]"/-->

    <!-- DY: migrated to IC-ISM-Functions.xsl -->
    <!--xsl:variable name="decomposableTetras"
        select="$decomposableTetraElems/catt:TetraToken/text()"/-->

    <!-- Either comment this line out or create a XSL that imports ISM-Rollup with only this line. It makes XSpec tests work
        Since XSpec does not handle global variables well but this not being global is SUPER slow.
        So you would only want this during XSpec as it would return to the very slow execution. -->
    <!--
        <xsl:variable name="ISM_RESOURCE_ELEMENT" select="/parent::*"/>
-->
    <xsl:variable name="resourceElement" select="util:resourceElementTraverse((root()))"/>

    <xd:doc>
        <xd:desc>
            <xd:p>Main template match from which all work starts. It is matching on the resource node for the document passed in since that is the only node
that is actually modified. All other nodes are passed through with identity transform.</xd:p>
        </xd:desc>
    </xd:doc>
    <xsl:template match="*[generate-id(.) = generate-id(util:resourceElement((root())))]">
        <xsl:variable name="classifiedDoc" as="xs:boolean">
            <xsl:value-of select="util:classifiedDoc()"/>
        </xsl:variable>

```

```

<xsl:variable name="resourceNodeContext" select="."/>
<xsl:copy>
  <xsl:apply-templates select="@* except @ism:*" />
  <xsl:apply-templates select="@ism:DESVersion | @ism:createDate | @ism:resourceElement | @ism:ownerProducer | @ism:compliesWith | @ism:ISMCATCESVersion" />
  <xsl:call-template name="addISMderived">
    <xsl:with-param name="derivativelyClassifiedBy" select="$derivativelyClassifiedBy" />
    <xsl:with-param name="derivedFrom" select="$derivedFrom" />
  </xsl:call-template>
  <xsl:call-template name="addISMclassification" />
  <xsl:call-template name="AddISMatomicEnergyMarkings" />
  <xsl:call-template name="AddISMnonUSControls" />
  <xsl:call-template name="AddISMFGI" />

  <xsl:call-template name="AddISMSCIcontrols" />
  <xsl:call-template name="AddISMhasApproximateMarkings" />
  <xsl:call-template name="AddISMdisseminationControls">
    <xsl:with-param name="ResourceNodeContext" select="$resourceNodeContext" />
  </xsl:call-template>
  <xsl:call-template name="AddISMdeclass">
    <xsl:with-param name="ResourceNodeContext" select="$resourceNodeContext" />
  </xsl:call-template>
  <xsl:call-template name="AddNTK">
    <xsl:with-param name="ResourceNodeContext" select="$resourceNodeContext" />
  </xsl:call-template>
  <xsl:call-template name="AddNotices">
    <xsl:with-param name="ResourceNodeContext" select="$resourceNodeContext" />
  </xsl:call-template>
  <xsl:apply-templates select="node() except ntk:Access" />
</xsl:copy>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM disseminationControls, and nonICmarkings attributes based on values in the document.</xd:p>
    <xd:p>Handles the following disseminationControls special cases</xd:p>
    <xd:ul>
      <xd:li>FOUO drops from classified Doc</xd:li>
      <xd:li>FOUO drops from DSEN Doc</xd:li>
      <xd:li>nonICmarkings SBU-NF adds NF in classified Doc</xd:li>
      <xd:li>nonICmarkings LES-NF adds NF in classified Doc</xd:li>
      <xd:li>NF anywhere drops REL, EYES, RELIDO, and DisplayOnly</xd:li>
    </xd:ul>
    <xd:p>Handles the following nonICmarkings special cases</xd:p>
    <xd:ul>
      <xd:li>SBU-NF changes to SBU in a classified Doc or when NF is already present</xd:li>
      <xd:li>LES-NF changes to LES in a classified Doc or when NF is already present</xd:li>
    </xd:ul>
  </xd:desc>
  <xd:param name="ResourceNodeContext">
    <xd:p>The resource node for the document or fragment being rolled up. </xd:p>
  </xd:param>
</xd:doc>
<xsl:template name="AddISMdisseminationControls">
  <xsl:param name="ResourceNodeContext" required="yes" />

```

```

<xsl:variable name="distinctDisseminationControls"
  select="distinct-values(//*[util:contributesToRollup(.) and @ism:disseminationControls]/xs:NMTOKENS(@ism:disseminationControls))"/>
<xsl:variable name="distinctnonICMarkings"
  select="distinct-values(//*[util:contributesToRollup(.) and @ism:nonICMarkings]/xs:NMTOKENS(@ism:nonICMarkings))"/>

<xsl:variable name="allOCPortions"
  select="//*[util:contributesToRollup(.) and xs:NMTOKENS(@ism:disseminationControls)='OC']"/>
<xsl:variable name="OCWithUSGov"
  select="$allOCPortions[some $dissem in xs:NMTOKENS(@ism:disseminationControls) satisfies $dissem ='OC-USGOV']"/>
<xsl:variable name="OCWithOutUSGov"
  select="$allOCPortions[every $dissem in xs:NMTOKENS(@ism:disseminationControls) satisfies $dissem !='OC-USGOV']"/>
<xsl:variable name="DropOCUSGOV" as="xs:boolean">
  <xsl:choose>
    <xsl:when test="count($OCWithUSGov) >0 and count($OCWithOutUSGov)>0">
      <xsl:sequence select="true()"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:sequence select="false()"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>

<xsl:if test="count($distinctDisseminationControls) != 0 or count($distinctnonICMarkings) != 0">
  <xsl:variable name="droppedWithNF"
    select="xs:NMTOKENS('REL DISPLAYONLY EYES RELIDO')"/>
  <xsl:variable name="NF" select="xs:NMTOKENS('NF')"/>
  <xsl:variable name="allPortionsRelido"
    as="xs:boolean"
    select="util:allDissemPortionsHave($ResourceNodeContext, $distinctDisseminationControls, xs:NMTOKENS('RELIDO'))"/>

  <xsl:variable name="allPortionsRelEyes"
    as="xs:boolean"
    select="util:allDissemPortionsHave($ResourceNodeContext, $distinctDisseminationControls, xs:NMTOKENS('REL EYES'))"/>

  <xsl:variable name="allPortionsRelEyesorDisplay"
    as="xs:boolean"
    select="util:allDissemPortionsHave($ResourceNodeContext, $distinctDisseminationControls, xs:NMTOKENS('REL EYES DISPLAYONLY'))"/>

  <xsl:variable name="anyPortionHasDSEN"
    select="$distinctDisseminationControls = xs:NMTOKENS('DSEN')"
    as="xs:boolean"/>

  <!-- It might be NF and not meet this variable but it can't be REL,EYES, or DISPLAYONLY if it meets these criteria -->
  <xsl:variable name="CertainNFDocument" as="xs:boolean">
    <xsl:choose>
      <xsl:when test="$distinctDisseminationControls = $NF">
        <xsl:copy-of select="true()"/>
      </xsl:when>
      <xsl:when test="not($allPortionsRelEyesorDisplay) and $distinctDisseminationControls =xs:NMTOKENS('REL EYES DISPLAYONLY')">
        <xsl:copy-of select="true()"/>
      </xsl:when>
      <xsl:when test="$distinctnonICMarkings = xs:NMTOKENS('LES-NF SBU-NF')">
        <xsl:copy-of select="true()"/>
      </xsl:when>
    </xsl:choose>
  </xsl:variable>

```



```

        <xsl:otherwise>
            <xsl:copy-of select="false()"/>
        </xsl:otherwise>
    </xsl:choose>

</xsl:variable>

<xsl:variable name="dissemFiltered1" as="xs:NMTOKEN*">
    <xsl:for-each select="$distinctDisseminationControls">
        <xsl:choose>
            <xsl:when test="not($allPortionsRelido) and (. = 'RELIDO')"/>
            <xsl:when test="$DropOCUSGOV and . = 'OC-USGOV' "/>
            <xsl:when test="$CertainNFDdocument and (. = 'REL' or . = 'DISPLAYONLY' or . = 'EYES' or . = 'RELIDO')"/>
            <xsl:when test="not($allPortionsRelEyes) and (. = 'REL' or . = 'EYES')"/>
            <xsl:when test="not($allPortionsRelEyesorDisplay) and (. = 'DISPLAYONLY')"/>
            <xsl:when test="$anyPortionHasDSEN and (. = 'FOUO')"/>
            <xsl:when test="util:classifiedDoc(root($ResourceNodeContext)) and not($anyPortionHasDSEN) and (. = 'FOUO')"/>

            <xsl:otherwise>
                <xsl:copy-of select="xs:NMTOKEN(current())"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
    <xsl:if test="$distinctDisseminationControls != $NF and $CertainNFDdocument">
        <xsl:copy-of select="xs:NMTOKEN('NF')"/>
    </xsl:if>
</xsl:variable>

<xsl:variable name="nonICmarkingsFiltered1" as="xs:NMTOKEN*">
    <xsl:for-each select="$distinctnonICmarkings">
        <xsl:choose>
            <xsl:when test="($distinctDisseminationControls = ('NF', 'REL', 'DISPLAYONLY') or util:classifiedDoc(root($ResourceNodeContext))) and (. = 'SBU-NF')">
                <xsl:copy-of select="xs:NMTOKEN('SBU')"/>
            </xsl:when>
            <xsl:when test="($distinctDisseminationControls = ('NF', 'REL', 'DISPLAYONLY') or util:classifiedDoc(root($ResourceNodeContext))) and (. = 'LES-NF')">
                <xsl:copy-of select="xs:NMTOKEN('LES')"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:copy-of select="xs:NMTOKEN(current())"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
</xsl:variable>
<xsl:if test="count($nonICmarkingsFiltered1) !=0">
    <xsl:attribute name="nonICmarkings" namespace="urn:us:gov:ic:ism">
        <xsl:value-of select="ism-func:join($nonICmarkingsFiltered1)"/>
    </xsl:attribute>
</xsl:if>

<xsl:variable name="DistinctRelPortions"
    select="distinct-values(//*[util:contributesToRollup(.) and @ism:releasableTo]/@ism:releasableTo)"/>
<xsl:variable name="RelCommonCountriesUnsorted"
    select="util:unsortedCommonRelTokens($DistinctRelPortions,count($DistinctRelPortions))"/>

```

```

<xsl:if test="count($dissemFiltered1) != 0">
  <xsl:variable name="dissemFiltered2" as="xs:NMTOKEN*">
    <xsl:choose>
      <xsl:when test="$dissemFiltered1 = xs:NMTOKENS('REL EYES')">
        <xsl:variable name="distinctReleasableTo"
          select="distinct-values(//*[util:contributesToRollup(.) and @ism:releasableTo]/xs:NMTOKENS(@ism:releasableTo))"/>
        <xsl:choose>
          <!-- There are common countries USA and something and at least on portion is REL drop EYES -->
          <xsl:when test="count($RelCommonCountriesUnsorted) > 1 and $dissemFiltered1 = xs:NMTOKENS('REL')">
            <xsl:copy-of select="$dissemFiltered1[not(. = xs:NMTOKENS('EYES'))]"/>
          </xsl:when>
          <!-- There are common countries USA and something and at no portion is REL keep EYES -->
          <xsl:when test="count($RelCommonCountriesUnsorted) > 1 and $dissemFiltered1 != xs:NMTOKENS('REL')">
            <xsl:copy-of select="$dissemFiltered1"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:copy-of select="$dissemFiltered1[not(. = xs:NMTOKENS('REL'))]"/>
            <xsl:copy-of select="xs:NMTOKENS('NF')"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy-of select="$dissemFiltered1"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:attribute name="disseminationControls" namespace="urn:us:gov:ic:ism">
    <xsl:value-of select="ism-func:sortDissemControlsPreCUI(ism-func:join($dissemFiltered2))"/>
  </xsl:attribute>

  <xsl:if test="$dissemFiltered2 = xs:NMTOKENS('REL EYES')">
    <xsl:attribute name="releasableTo" namespace="urn:us:gov:ic:ism">
      <xsl:value-of select="ism-func:sortReleaseto(ism-func:join($RelCommonCountriesUnsorted))"/>
    </xsl:attribute>
  </xsl:if>
</xsl:if>
</xsl:if>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM hasApproximateMarkings attribute based on values in the document.</xd:p>
    <xd:p>There are no known special cases to handle at this time.</xd:p>
  </xd:desc>
</xd:doc>
<xsl:template name="AddISMhasApproximateMarkings">
  <xsl:variable name="distincthasApproximateMarkings"
    select="distinct-values(//*[util:contributesToRollup(.) and @ism:hasApproximateMarkings]/@ism:hasApproximateMarkings)"/>
  <xsl:if test="(count($distincthasApproximateMarkings) != 0) and $distincthasApproximateMarkings = ('true', '1')">
    <xsl:attribute name="hasApproximateMarkings" namespace="urn:us:gov:ic:ism">
      <xsl:value-of select="true()"/>
    </xsl:attribute>
  </xsl:if>
</xsl:template>

```

```
</xsl:if>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM ism:NoticeList structure based on values in the document.
    Assumes using ARH and that resource node was the arh:security element structure.
  </xd:p>
  </xd:desc>
  <xd:param name="ResourceNodeContext">
    <xd:p>The resource node for the document or fragment being rolled up. </xd:p>
  </xd:param>
</xd:doc>
<xsl:template name="AddNotices">
  <xsl:param name="ResourceNodeContext" required="yes"/>

  <xsl:if test="//ism:NoticeList and $ResourceNodeContext/self::arh:Security">
    <xsl:variable name="tempNoticeList">
      <ism:NoticeList ism:classification="U"
        ism:resourceElement="true"
        ism:ownerProducer="USA">
        <!-- Notices have no variability so there is no need to normalize before distinct-deep compare. -->
        <xsl:variable name="distinctNotices" select="func:distinct-deep(//ism:Notice)"/>
        <xsl:copy-of select="$distinctNotices"/>
      </ism:NoticeList>
    </xsl:variable>
    <xsl:variable name="RollupDoneNoticeList">
      <xsl:apply-templates select="$tempNoticeList"/>
    </xsl:variable>
    <xsl:apply-templates select="$RollupDoneNoticeList" mode="stripBlockAttributes"/>
  </xsl:if>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM ntk:Access structure based on values in the document.
    Assumes using ARH and that resource node was the arh:security element structure.
  </xd:p>
  </xd:desc>
  <xd:param name="ResourceNodeContext">
    <xd:p>The resource node for the document or fragment being rolled up. </xd:p>
  </xd:param>
</xd:doc>
<xsl:template name="AddNTK">
  <xsl:param name="ResourceNodeContext" required="yes"/>

  <xsl:if test="//ntk:Access and $ResourceNodeContext/self::arh:Security">
    <xsl:variable name="tempNTKAccess">
      <ntk:Access ism:classification="U"
        ism:ownerProducer="USA"
        ism:resourceElement="true">
        <xsl:choose>
          <xsl:when test="count(//ntk:Access)=1">
```

```

        <!-- Only 1 NTK in the doc nothing to rollup just copy -->
        <xsl:copy-of select="//ntk:Access/ntk:*/">
            </xsl:when>
            <xsl:otherwise>
                <xsl:if test="//ntk:RequiresAllOf">
                    <xsl:call-template name="processNTKAllOf"/>
                </xsl:if>
                <xsl:if test="//ntk:RequiresAnyOf">
                    <xsl:call-template name="processNTKAnyOf"/>
                </xsl:if>
            </xsl:otherwise>
        </xsl:choose>
    </ntk:Access>
</xsl:variable>

    <xsl:variable name="RollupDonetempNTKAccess">
        <xsl:apply-templates select="$tempNTKAccess"/>
    </xsl:variable>
    <xsl:apply-templates select="$RollupDonetempNTKAccess" mode="stripBlockAttributes"/>
</xsl:if>
</xsl:template>

<xsl:template name="processNTKAllOf">
    <xsl:if test="count(//ntk:Access//ntk:RequiresAllOf/*[not(self::ntk:AccessProfileList)])>1 ">
        <!-- OLD probably 2013 NTK we can't cope. -->
        <ntk:NTK-13-unsupported>There were NTK structures that pre 2015 and rollup was not defined to handle them.</ntk:NTK-13-unsupported>
    </xsl:if>
    <ntk:RequiresAllOf>
        <ntk:AccessProfileList>
            <!-- NTK need to normalize before distinct-deep compare. -->
            <xsl:variable name="normalizedNTKProfiles">
                <xsl:apply-templates mode="normalizeNTKProfile"
                    select="//ntk:RequiresAllOf/ntk:AccessProfileList/ntk:AccessProfile"/>
            </xsl:variable>
            <xsl:variable name="distinctNTKProfiles"
                select="functx:distinct-deep($normalizedNTKProfiles/ntk:AccessProfile)"/>

            <xsl:copy-of select="$distinctNTKProfiles"/>

        </ntk:AccessProfileList>
    </ntk:RequiresAllOf>
</xsl:template>

<xsl:template name="processNTKAnyOf">
    <xsl:variable name="normalizedNTKAnyOfProfiles">
        <xsl:apply-templates mode="normalizeNTKProfile"
            select="//ntk:RequiresAnyOf/ntk:AccessProfileList/ntk:AccessProfile"/>
    </xsl:variable>
    <xsl:variable name="distinctNTKAnyOfProfiles"
        select="functx:distinct-deep($normalizedNTKAnyOfProfiles/ntk:AccessProfile)"/>

    <xsl:variable name="normalizedNTKAnyOfAccessGroup">
        <xsl:apply-templates mode="normalizeNTKProfile"
            select="//ntk:RequiresAnyOf/ntk:AccessGroupList/ntk:AccessGroup"/>

```

don't

input.-->

```

    </xsl:variable>
    <xsl:variable name="distinctNTKAnyOfAccessGroup"
        select="functx:distinct-deep($normalizedNTKAnyOfAccessGroup/ntk:AccessGroup)"/>

    <xsl:choose>
        <xsl:when test="count(//ntk:Access//ntk:RequiresAnyOf)>1 and count($distinctNTKAnyOfProfiles)=1 and count($distinctNTKAnyOfAccessGroup)=0">
            <!-- There is exactly 1 anyOf profile use it and be happy. There may be some whose ntk profile has A or A a somewhat pointless anyOf but we don't
                care. There also was not any NTK 2013 group element to worry. -->
            <ntk:RequiresAnyOf>
                <ntk:AccessProfileList>
                    <xsl:copy-of select="($distinctNTKAnyOfProfiles)"/>
                </ntk:AccessProfileList>
            </ntk:RequiresAnyOf>
        </xsl:when>
        <xsl:when test="count(//ntk:Access//ntk:RequiresAnyOf)>1 and count($distinctNTKAnyOfProfiles)=0 and count($distinctNTKAnyOfAccessGroup)=1">
            <!-- There is exactly 1 anyOf AccessGroup use it and be happy even if it is from NTK 2013. There may be some whose ntk AccessGroup has A or A a somewhat pointless anyOf but we
                care. There also was not any NTK profile element to worry. -->
            <!-- NOTE that since this appears to be NTK from 2013 or earlier the ntk:RequiresAnyOf had ISM attributes on it. ISM 2015 and later do not. -->
            <ntk:RequiresAnyOf ism:classification="U" ism:ownerProducer="USA">
                <ntk:AccessGroupList>
                    <xsl:copy-of select="($distinctNTKAnyOfAccessGroup)"/>
                </ntk:AccessGroupList>
            </ntk:RequiresAnyOf>
        </xsl:when>
        <xsl:when test="count(//ntk:Access//ntk:RequiresAnyOf)>1 and ( count($distinctNTKAnyOfProfiles)>1 or count($distinctNTKAnyOfAccessGroup)>1) ">
            <!-- More than 1 ntk:RequiresAnyOf we can't do rollup. Cause a bad NTK structure to fail validation. This is an intentional invalid marking to cause validation to fail on bad
                input.-->
            <ntk:TooManyRequiresAnyOf>There were more than 1 RequiresAnyOf rollup not possible</ntk:TooManyRequiresAnyOf>
        </xsl:when>
        <xsl:when test="count(//ntk:Access//ntk:RequiresAnyOf/*[not(self::ntk:AccessProfileList)])>1 ">
            <!-- OLD probably 2013 NTK with multiple anyOF we can't cope.-->
            <ntk:NTK-13-unsupported>There were more than 1 RequiresAnyOf and they were pre 2015 so rollup not possible</ntk:NTK-13-unsupported>
        </xsl:when>

        <xsl:otherwise>
            <!-- Copy the 1 distinct ntk:RequiresAnyOf if it exists -->
            <xsl:copy-of select="(//ntk:Access//ntk:RequiresAnyOf)[1]"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM declassDate attribute based on values in the document.</xd:p>
    <xd:p>Handles the following special cases</xd:p>
    <xd:ul>
      <xd:li>If there are RD,FRD, or TFNI AEA markings add the AEA exception.</xd:li>
      <xd:li>If there are any NATO/NATO:NAC portion or NATO/NATO:NAC FGI add the NATO exception.</xd:li>
    </xd:ul>
  </xd:desc>
  <xd:param name="ResourceNodeContext">
    <xd:p>The resource node for the document or fragment being rolled up. </xd:p>
  </xd:param>
</xd:doc>
```

```

</xd:doc>
<xsl:template name="AddISMdeclass">
  <xsl:param name="ResourceNodeContext" required="yes"/>
  <xsl:if test="util:classifiedDoc(root($ResourceNodeContext))">
    <xsl:variable name="distinctdeclassDate"
      select="distinct-values(//*[util:contributesToRollup(.) and @ism:declassDate]/@ism:declassDate)"
      as="xs:date"/>
    <xsl:variable name="distinctdeclassEvent"
      select="distinct-values(//*[util:contributesToRollup(.) and @ism:declassEvent]/@ism:declassEvent)"/>
    <xsl:variable name="distinctdeclassException"
      select="distinct-values(//*[util:contributesToRollup(.) and @ism:declassException]/xs:NMTOKENS(@ism:declassException))"/>

    <xsl:variable name="countNATO"
      as="xs:integer"
      select="count(//*[util:contributesToRollup(.) and (starts-with(@ism:ownerProducer, 'NATO') or contains(@ism:FGISourceOpen, 'NATO'))])"/>
    <xsl:variable name="countAEA"
      as="xs:integer"
      select="count(//*[util:contributesToRollup(.) and @ism:atomicEnergyMarkings and @ism:classification != 'U'])"/>
    <xsl:variable name="exceptions" as="xs:NMTOKEN*"
      <xsl:choose>
        <xsl:when test="$countAEA != 0 and $countNATO != 0">
          <xsl:copy-of select="xs:NMTOKEN('NATO-AEA')"/>
        </xsl:when>
        <xsl:when test="$countNATO != 0">
          <xsl:copy-of select="xs:NMTOKEN('NATO')"/>
        </xsl:when>
        <xsl:when test="$countAEA != 0">
          <xsl:copy-of select="xs:NMTOKEN('AEA')"/>
        </xsl:when>
      </xsl:choose>
    <xsl:if test="count($distinctdeclassException) > 0">
      <xsl:copy-of select="$distinctdeclassException"/>
    </xsl:if>
  </xsl:variable>

  <xsl:if test="count(distinct-values($exceptions)) != 0 ">
    <xsl:attribute name="declassException" namespace="urn:us:gov:ic:ism">
      <xsl:value-of select="ism-func:join(distinct-values($exceptions))"/>
    </xsl:attribute>
  </xsl:if>
  <xsl:if test="count($distinctdeclassEvent) != 0">
    <xsl:attribute name="declassEvent" namespace="urn:us:gov:ic:ism">
      <xsl:value-of select="ism-func:join($distinctdeclassEvent)"/>
    </xsl:attribute>
  </xsl:if>
  <xsl:choose>
    <xsl:when test="count($distinctdeclassDate) != 0 and (count($exceptions) = 0 or not(normalize-space($exceptions)))">
      <xsl:variable name="maxDate" select="max($distinctdeclassDate)"/>
      <xsl:attribute name="declassDate" namespace="urn:us:gov:ic:ism">
        <xsl:value-of select="string($maxDate)"/>
      </xsl:attribute>
    </xsl:when>
    <xsl:when test="count($exceptions) = 0 ">
      <xsl:attribute name="declassDate" namespace="urn:us:gov:ic:ism">

```



```

        <xsl:value-of select="format-date(current-date() + xs:yearMonthDuration('P25Y'),'[Y0001]-[M01]-[D01]')"/>
      </xsl:attribute>
    </xsl:when>
    <xsl:otherwise/>
  </xsl:choose>
</xsl:if>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM FGIsorceOpen, or FGIsorceProtected attribute based on values in the document. Assumes US document</xd:p>
    <xd:p>Handles the following disseminationControls special cases</xd:p>
    <xd:ul>
      <xd:li>Any FGIsorceProtected supress all FGIsorceOpen</xd:li>
      <xd:li>Any @ism:ownerProducer not USA are added to FGIsorceOpen</xd:li>
    </xd:ul>
  </xd:desc>
</xd:doc>
<xsl:template name="AddISMFGI">
  <xsl:variable name="distinctFGIsorceOpen"
    select="distinct-values(//*[util:contributesToRollup(.) and @ism:FGIsorceOpen]/xs:NMTOKENS(@ism:FGIsorceOpen))"/>
  <xsl:variable name="distinctFGIsorceProtected"
    select="distinct-values(//*[util:contributesToRollup(.) and @ism:FGIsorceProtected]/xs:NMTOKENS(@ism:FGIsorceProtected))"/>
  <xsl:variable name="distinctownerProducer"
    select="distinct-values(//*[util:contributesToRollup(.) and @ism:ownerProducer != 'USA']/xs:NMTOKENS(@ism:ownerProducer))"/>

  <xsl:choose>
    <xsl:when test="(count($distinctFGIsorceProtected) != 0)">
      <xsl:attribute name="FGIsorceProtected" namespace="urn:us:gov:ic:ism">FGI</xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="FGIUnion" as="xs:anyAtomicType">
        <xsl:sequence select="distinct-values(($distinctFGIsorceOpen, $distinctownerProducer[not(. = xs:NMTOKENS('USA'))]))"/>
      </xsl:variable>
      <xsl:if test="count($FGIUnion) != 0">
        <xsl:attribute name="FGIsorceOpen" namespace="urn:us:gov:ic:ism">
          <xsl:value-of select="ism-func:sortFGIOpen(ism-func:join($FGIUnion))"/>
        </xsl:attribute>
      </xsl:if>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM nonUSControls attribute based on values in the document.</xd:p>
    <xd:p>There are no known special cases to handle at this time.</xd:p>
  </xd:desc>
</xd:doc>
<xsl:template name="AddISMnonUSControls">
  <xsl:variable name="distinctnonUSControls"
    select="distinct-values(//*[util:contributesToRollup(.) and @ism:nonUSControls]/xs:NMTOKENS(@ism:nonUSControls))"/>
  <xsl:if test="count($distinctnonUSControls) != 0">
    <xsl:attribute name="nonUSControls" namespace="urn:us:gov:ic:ism">

```

```

        <xsl:value-of select="ism-func:join($distinctnonUSControls)"/>
      </xsl:attribute>
    </xsl:if>
  </xsl:template>

  <xd:doc>
    <xd:desc>
      <xd:p>Create an ISM classification attribute based on values in the document.</xd:p>
      <xd:p>Handles the following special cases</xd:p>
      <xd:ul>
        <xd:li>R gets upgraded to C</xd:li>
        <xd:li>Lack of any valid token gets classification set to "InvalidClassificationInput" this is an intentional invalid marking to cause validation to fail on bad
input. </xd:li>
      </xd:ul>
    </xd:desc>
  </xd:doc>
  <xsl:template name="addISMclassification">
    <xsl:attribute name="classification" namespace="urn:us:gov:ic:ism">
      <xsl:choose>
        <xsl:when test="key('classification', 'TS')">TS</xsl:when>
        <xsl:when test="key('classification', 'S')">S</xsl:when>
        <xsl:when test="key('classification', 'C')">C</xsl:when>
        <xsl:when test="key('classification', 'R')">C</xsl:when>
        <xsl:when test="key('classification', 'U')">U</xsl:when>
        <xsl:otherwise>InvalidClassificationInput</xsl:otherwise>
      </xsl:choose>
    </xsl:attribute>
  </xsl:template>

  <xd:doc>
    <xd:desc>
      <xd:p>Create an ISM SCIcontrols attribute based on values in the document. If there are no SCIcontrols don't create the attribute.</xd:p>
      <xd:p>There are no special cases for SCI just the distinct union of all sorted alphabetically</xd:p>
    </xd:desc>
  </xd:doc>
  <xsl:template name="AddISMSCIcontrols">
    <xsl:variable name="distinctSCI"
      select="distinct-values(//*[util:contributesToRollup(.) and @ism:SCIcontrols]/xs:NMTOKENS(@ism:SCIcontrols))"
      as="xs:NMTOKEN*" />
    <xsl:if test="count($distinctSCI) !=0">
      <xsl:attribute name="SCIcontrols" namespace="urn:us:gov:ic:ism">
        <xsl:value-of select="ism-func:sortSciControls(ism-func:join($distinctSCI))"/>
      </xsl:attribute>
    </xsl:if>
  </xsl:template>

  <xd:doc>
    <xd:desc>
      <xd:p>Create an ISM atomicEnergyMarkings attribute based on values in the document. If there are no atomicEnergyMarkings don't create the
attribute.</xd:p>
      <xd:p>Handles the following special cases</xd:p>
      <xd:ul>
        <xd:li>UCNI and DCNI are dropped from the banner of classified documents.</xd:li>
      </xd:ul>
    </xd:desc>
  </xd:doc>

```



```

    </xd:desc>
</xd:doc>
<xsl:template name="AddISMatomicEnergyMarkings">
  <xsl:variable name="distinctAtomic"
    select="distinct-values(//*[util:contributesToRollup(.) and @ism:atomicEnergyMarkings]/xs:NMTOKENS(@ism:atomicEnergyMarkings))"/>
  <xsl:variable name="unclassAtomic" select="xs:NMTOKENS('UCNI DCNI')"/>

  <xsl:variable name="atomicFiltered1">
    <xsl:choose>
      <xsl:when test="(key('classification', 'TS') | key('classification', 'S') | key('classification', 'C') | key('classification', 'R'))">
        <xsl:copy-of select="$distinctAtomic[not(. = $unclassAtomic)]"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy-of select="$distinctAtomic"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:if test="normalize-space($atomicFiltered1)">
    <xsl:attribute name="atomicEnergyMarkings" namespace="urn:us:gov:ic:ism">
      <xsl:value-of select="ism-func:sortAtomicenergymarkings($atomicFiltered1)"/>
    </xsl:attribute>
  </xsl:if>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Create an ISM @ism:derivativelyClassifiedBy and @ism:derivedFrom attributes based partly on values in the document and partly on parameters
    passed in.</xd:p>
    <xd:p>Handles the following special cases</xd:p>
    <xd:ul>
      <xd:li>Unclassified documents don't get these attributes</xd:li>
      <xd:li>Classified documents get them set to the parameters passed.</xd:li>
    </xd:ul>
  </xd:desc>
  <xd:param name="derivedFrom">
    <xd:p>The source that should be cited in a derived from statement.</xd:p>
  </xd:param>
  <xd:param name="derivativelyClassifiedBy">
    <xd:p>The Person who is responsible for the decision. Since this is software it should be the accrediting human responsible for it being in
    production.</xd:p>
  </xd:param>
</xd:doc>
<xsl:template name="addISMderived">
  <xsl:param name="derivedFrom" required="yes"/>
  <xsl:param name="derivativelyClassifiedBy" required="yes"/>
  <xsl:if test="util:classifiedDoc()">
    <xsl:attribute name="derivativelyClassifiedBy"
      namespace="urn:us:gov:ic:ism"
      select="$derivativelyClassifiedBy"/>
    <xsl:attribute name="derivedFrom"
      namespace="urn:us:gov:ic:ism"
      select="$derivedFrom"/>
  </xsl:if>

```

```

</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Identity Transform uses mode="#current" to preserve whatever mode was used.</xd:p>
  </xd:desc>
</xd:doc>
<xsl:template match="@* | node()" mode="#default stripBlockAttributes identity">
  <xsl:copy>
    <xsl:apply-templates select="@* | node()" mode="#current"/>
  </xsl:copy>
</xsl:template>

<xd:doc>
  <xd:desc>
    <xd:p>Strip out block attributes for things like notice list that needed to have a rollup done but are not really resource nodes.</xd:p>
  </xd:desc>
</xd:doc>
<xsl:template match="@ism:resourceElement | @ism:derivedFrom | @ism:derivativelyClassifiedBy | @ism:declassDate"
  mode="stripBlockAttributes"
  priority="10"/>

<xd:doc>
  <xd:desc>Normalize an ntk:AccessProfile by sorting all of the ntk:VocabularyType followed by all the ntk:AccessProfileValue</xd:desc>
</xd:doc>
<xsl:template match="ntk:AccessProfile" mode="normalizeNTKProfile">
  <xsl:copy>
    <xsl:apply-templates select="@*" mode="identity"/>
    <xsl:apply-templates select="ntk:AccessPolicy | ntk:ProfileDes" mode="identity"/>
    <xsl:apply-templates select="ntk:VocabularyType" mode="identity">
      <xsl:sort select="./@ntk:name" data-type="text" order="ascending"/>
    </xsl:apply-templates>
    <xsl:apply-templates select="ntk:AccessProfileValue" mode="identity">
      <xsl:sort select="concat(./@ntk:qualifier,@ntk:vocabulary,string(.)) "
        data-type="text"
        order="ascending"/>
    </xsl:apply-templates>
  </xsl:copy>
</xsl:template>

<xd:doc>
  <xd:desc>Normalize an ntk:AccessGroup by sorting all of the ntk:AccessGroupValue</xd:desc>
</xd:doc>
<xsl:template match="ntk:AccessGroup" mode="normalizeNTKProfile">
  <xsl:copy>
    <xsl:apply-templates select="@*" mode="identity"/>
    <xsl:apply-templates select="ntk:AccessPolicy | ntk:ProfileDes" mode="identity"/>
    <xsl:apply-templates select="ntk:AccessGroupValue" mode="identity">
      <xsl:sort select="." data-type="text" order="ascending"/>
    </xsl:apply-templates>
  </xsl:copy>
</xsl:template>

<!--*****-->

```

```

<!-- (U) Custom XSLT functions -->
<!--*****-->

<xd:doc>
  <xd:desc>
    <xd:p>Returns true if the attribute @ism:excludeFromRollup is not present or does not evaluate to 'true'</xd:p>
  </xd:desc>
  <xd:param name="context"/>
</xd:doc>
<xsl:function name="util:contributesToRollup" as="xs:boolean">
  <xsl:param name="context"/>
  <xsl:sequence select="
    not($context/@ism:excludeFromRollup castable as xs:boolean and $context/@ism:excludeFromRollup = true()) and $context/
@ism:ownerProducer
    and not(generate-id($context) =generate-id(util:resourceElement($context)) )"/>
</xsl:function>

<xd:doc>
  <xd:desc>
    <xd:p>Returns true() if the document contains any classified portions that contribute to rollup.</xd:p>
  </xd:desc>
  <xd:param name="context">
    <xd:p>used by the Key function since this is in an xsl:function it's required. Just pass in "/"</xd:p>
  </xd:param>
</xd:doc>
<xsl:function name="util:classifiedDoc" as="xs:boolean">
  <xsl:param name="context"/>
  <xsl:choose>
    <xsl:when test="key('classification', 'TS', $context)">
      <xsl:value-of select="true()"/>
    </xsl:when>
    <xsl:when test="key('classification', 'S', $context)">
      <xsl:value-of select="true()"/>
    </xsl:when>
    <xsl:when test="key('classification', 'C', $context)">
      <xsl:value-of select="true()"/>
    </xsl:when>
    <xsl:when test="key('classification', 'R', $context)">
      <xsl:value-of select="true()"/>
    </xsl:when>
    <xsl:when test="key('classification', 'U', $context)">
      <xsl:value-of select="false()"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message select="'Classification unknown value'"/>
      <xsl:value-of select="true()"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:function>

<xd:doc>
  <xd:desc>
    <xd:p>Checks that every portion has a particular Dissem Control.</xd:p>
  </xd:desc>
  <xd:param name="context"/>
  <xd:param name="distinctDisseminationControls"/>

```

```

    <xd:param name="Dissem"/>
</xd:doc>
<xsl:function name="util:allDissemPortionsHave" as="xs:boolean">
  <xsl:param name="context"/>
  <xsl:param name="distinctDisseminationControls"/>
  <xsl:param name="Dissem" as="xs:NMTOKEN*"/>
  <xsl:choose>
    <xsl:when test="$distinctDisseminationControls = $Dissem">
      <xsl:variable name="contributescount"
        select="count(root($context)//*[util:contributesToRollup(.) and util:PartakesInFDR(.)])"/>
      <xsl:variable name="contributesSpecifiedcount"
        select="count(root($context)//*[util:contributesToRollup(.) and util:PartakesInFDR(.)][normalize-space(@ism:disseminationControls)
[xs:NMTOKENS(@ism:disseminationControls) = $Dissem]])"/>
      <xsl:choose>
        <xsl:when test="$contributesSpecifiedcount = $contributescount">
          <xsl:value-of select="true()"/>
        </xsl:when>
        <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>

<xd:doc>
  <xd:desc> Given the a set of Rel strings return hash of those tokens count in the document </xd:desc>
  <xd:param name="relToStrings"/>
</xd:doc>
<xsl:function name="util:createRelTokenHash" as="item()*">
  <xsl:param name="relToStrings" as="xs:string*"/>
  <xsl:for-each-group select="for $each in $relToStrings return distinct-values(ism-func:tokenize($each))"
    group-by=".">
    <xsl:sort select="current-grouping-key()"/>
    <token count="{count(current-group())}">
      <xsl:value-of select="current-grouping-key()"/>
    </token>
  </xsl:for-each-group>
</xsl:function>

<xd:doc>
  <xd:desc>
    <xd:p>Cover function to allow calling without a count of portions.</xd:p>
  </xd:desc>
  <xd:param name="relToStrings"/>
</xd:doc>
<xsl:function name="util:commonToAllTokens">
  <xsl:param name="relToStrings" as="xs:string*"/>
  <xsl:sequence select="util:commonToAllTokens($relToStrings,$countFdrPortions)"/>
</xsl:function>

```

```

<xd:doc>
  <xd:desc>
    <xd:p>Determine what values are common to all portions for a REL to logic.</xd:p>
  </xd:desc>
  <xd:param name="relToStrings"/>
  <xd:param name="countFdrPortionsLocal"/>
</xd:doc>
<xsl:function name="util:commonToAllTokens">
  <xsl:param name="relToStrings" as="xs:string*" />
  <xsl:param name="countFdrPortionsLocal" as="xs:integer" />
  <xsl:variable name="RelTokenHash" select="util:createRelTokenHash($relToStrings)" />
  <xsl:sequence select="$RelTokenHash[@count=$countFdrPortionsLocal]" />
</xsl:function>

<xd:doc>
  <xd:desc> Remove the tokens that are common to all rel strings leaving us with the tokens that need expanding or are unmatched countries. </xd:desc>
  <xd:param name="relToStrings"/>
  <xd:param name="countFdrPortionsLocal"/>
</xd:doc>
<xsl:function name="util:RelStringsWithoutCommonTokens">
  <xsl:param name="relToStrings" as="xs:string*" />
  <xsl:param name="countFdrPortionsLocal" as="xs:integer" />
  <xsl:variable name="CommonTokens"
    select="util:commonToAllTokens($relToStrings,$countFdrPortionsLocal)" />
  <xsl:for-each-group select="for $each in $relToStrings return ism-func:join(distinct-values(for $relToken in ism-func:tokenize($each) return (if (some $Token in
$CommonTokens satisfies $Token=$relToken) then ' ' else ism-func:padValue($relToken) )))"
    group-by=".">
    <xsl:sequence select="string(current-grouping-key())" />
  </xsl:for-each-group>
</xsl:function>

<xd:doc>
  <xd:desc>
    <xd:p>Return the common tokens without any particular order.</xd:p>
  </xd:desc>
  <xd:param name="relToStrings"/>
  <xd:param name="countFdrPortionsLocal"/>
</xd:doc>
<xsl:function name="util:unsortedCommonRelTokens">
  <xsl:param name="relToStrings" as="xs:string*" />
  <xsl:param name="countFdrPortionsLocal" as="xs:integer" />
  <xsl:variable name="CommonTokens"
    select="util:commonToAllTokens($relToStrings,$countFdrPortionsLocal)" />
  <xsl:variable name="relToMinusCommonTokens"
    select="util:RelStringsWithoutCommonTokens($relToStrings,$countFdrPortionsLocal)" />
  <xsl:variable name="expandTetrasForRelTokensNotCommon"
    select="util:expandAllTetras($relToMinusCommonTokens)" />
  <xsl:variable name="CommonTokensAfterExpansion"
    select="util:commonToAllTokens($expandTetrasForRelTokensNotCommon,$countFdrPortionsLocal)" />
  <xsl:sequence select="$CommonTokens | $CommonTokensAfterExpansion" />
</xsl:function>

<!-- Maybe odd stuff lifted from ISM -->

```

```
<xd:doc>
  <xd:desc> 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 ism-func:expandDecomposableTetras() </xd:desc>
  <xd:param name="relToStrings" />
</xd:doc>
<xsl:function name="util:expandAllTetras" as="xs:string*">
  <xsl:param name="relToStrings" as="xs:string*" />

  <xsl:variable name="allTokens" as="xs:string*">
    <xsl:for-each select="$relToStrings">
      <xsl:variable name="expandedCountryTokens"
        select="ism-func:expandDecomposableTetras(.)" />
      <xsl:value-of select="ism-func:padValue(ism-func:join($expandedCountryTokens))" />
    </xsl:for-each>
  </xsl:variable>

  <xsl:sequence select="$allTokens" />
</xsl:function>

<xd:doc>
  <xd:desc> Recursively remove all decomposable tetragraphs in the given $relTo string
  and replace them with their constituent countries. Note: Does not include USA </xd:desc>
  <xd:param name="relTo" />
</xd:doc>
<!-- DY: refactored into IC-ISM-Functions.xsl -->
<!--xsl:function
  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)[. = $decomposableTetras][1]" />
        <xsl:variable name="currTetraCountries"
          select="ism-func:join(util:getCountriesForTetra($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))[. != 'USA']" />
</xsl:function-->

<xd:doc>
```

```

        <xd:desc> Returns true if the given $relTo string (e.g. 'USA CAN GBR') contains any
        tetragraphs that can be decomposed into its constituent countries </xd:desc>
        <xd:param name="relTo"/>
    </xd:doc>
    <!-- DY: refactored into IC-ISM-Functions.xsl -->
<!--xsl:function
    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-->

<xd:doc>
    <xd:desc>
        Returns true if any token in the attribute value matches at least one token in the provided list.
    </xd:desc>
    <xd:param name="attribute"/>
    <xd:param name="tokenList"/>
</xd:doc>
    <!-- DY: refactored into IC-ISM-Functions.xsl -->
<!--xsl:function
    name="util:containsAnyOfTheTokens"
    as="xs:boolean">
    <xsl:param name="attribute"/>
    <xsl:param name="tokenList" as="xs:string*"/>
    <xsl:sequence select="some $attrToken in tokenize(normalize-space(string($attribute)), ' ') satisfies $attrToken = $tokenList"/>
</xsl:function-->

<xd:doc>
    <xd:desc> Returns the sequence of country codes that correspond to the given $tetra </xd:desc>
    <xd:param name="tetra"/>
</xd:doc>
    <!-- DY: refactored into IC-ISM-Functions.xsl -->
<!--xsl:function
    name="util:getCountriesForTetra"
    as="xs:string*">
    <xsl:param name="tetra" as="xs:string"/>

    <xsl:sequence select="$decomposableTetraElems[catt:TetraToken/text() = $tetra]/catt:Membership/*/text()"/>
</xsl:function-->

<xd:doc>
    <xd:desc> Returns normalized $value with a preceding and subsequent space (' ') character </xd:desc>
    <xd:param name="value"/>
</xd:doc>
    <!-- DY: refactored into IC-ISM-Functions.xsl -->
<!--xsl:function
    name="util:padValue"
    as="xs:string">
    <xsl:param name="value" as="xs:string?"/>

    <xsl:value-of select="concat(' ', normalize-space($value), ' ')" />
</xsl:function-->
```



```
<xd:doc>
    <xd:desc> Returns the given $value with its values broken into tokens using whitespace as delimiters </xd:desc>
    <xd:param name="value"/>
</xd:doc>
<!-- DY: refactored into IC-ISM-Functions.xsl -->
<!--xsl:function
    name="util:tokenize"
    as="xs:string*">
    <xsl:param name="value" as="xs:string?"/>

    <xsl:sequence select="tokenize(normalize-space($value), ' ')" />
</xsl:function-->

<xd:doc>
    <xd:desc>
        Accepts an element.
        Returns true if the element contains any Foreign Disclosure & Release (FD&R) markings; false otherwise.
    </xd:desc>
    <xd:param name="elementNode"/>
</xd:doc>
<xsl:function name="util:containsFDR" as="xs:boolean">
    <xsl:param name="elementNode" as="node()"/>
    <xsl:sequence select="$elementNode/@ism:releasableTo or $elementNode/@ism:displayOnlyTo or ism-func:containsAnyOfTheTokens($elementNode/@ism:disseminationControls, ('NF',
'RELIDO')) or ism-func:containsAnyOfTheTokens($elementNode/@ism:nonICmarkings, ('LES-NF', 'SBU-NF'))" />
</xsl:function>

<xd:doc>
    <xd:desc>
        Returns true if the element would partake in an Foreign Disclosure & Release (FD&R) rollup decision.
    </xd:desc>
    <xd:param name="elementNode"/>
</xd:doc>
<xsl:function name="util:PartakesInFDR" as="xs:boolean">
    <xsl:param name="elementNode" as="node()"/>
    <xsl:choose>
        <xsl:when test="$elementNode[@ism:classification = 'U']">
            <xsl:choose>
<!-- Uncaveated unclassified information does not impact FDR rollup. -->
                <xsl:when test="$elementNode[not(@ism:disseminationControls) and not(@ism:nonICmarkings)]">
                    <xsl:value-of select="false()" />
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="true()" />
                </xsl:otherwise>
            </xsl:choose>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="true()" />
        </xsl:otherwise>
    </xsl:choose>
</xsl:function>
```



```
<xd:doc>
  <xd:desc> Return all the tags that participate in rollup decisions.</xd:desc>
  <xd:param name="context"/>
</xd:doc>
<xsl:function name="util:partTags">
  <xsl:param name="context"/>
  <xsl:sequence select="$context/descendant-or-self::node()[@ism:classification and util:contributesToRollup(.)]"/>
</xsl:function>

<xd:doc>
  <xd:desc>
    Count how many of partTags participate in Foreign Disclosure & Release (FD&R) decisions.
  </xd:desc>
  <xd:param name="context"/>
</xd:doc>
<xsl:function name="util:countFdrPortions">
  <xsl:param name="context"/>
  <xsl:value-of select="count(util:partTags($context)[util:PartakesInFDR(.)])"/>
</xsl:function>

<xd:doc>
  <xd:desc>
    <xd:p>The first element in document order that has resourceElement = true</xd:p>
  </xd:desc>
  <xd:param name="context"/>
</xd:doc>
<xsl:function name="util:resourceElement">
  <xsl:param name="context"/>
  <xsl:choose>
    <xsl:when test="exists($resourceElement)">
      <xsl:sequence select="$resourceElement"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:sequence select="util:resourceElementTraverse($context)"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:function>

<xsl:function name="util:resourceElementTraverse">
  <xsl:param name="context"/>
  <xsl:sequence select="(root($context)//*[@ism:resourceElement castable as xs:boolean and @ism:resourceElement = true() ] ) [1]"/>
</xsl:function>
</xsl:stylesheet>
```

2.3 - ISM-Rollup-forXSpec.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl"
                xmlns:util="urn:us:gov:ic:ism-rollup:xsl:util"
                exclude-result-prefixes="xs xd"
                version="2.0">

  <xsl:import href="ISM-Rollup.xsl"/>

  <xd:doc scope="stylesheet">
    <xd:desc>
      <xd:p>
        <xd:b>Created on:</xd:b> Jul 26, 2020</xd:p>
      <xd:p>
        <xd:b>Author:</xd:b> bob</xd:p>
      <xd:p>Overrides the resourceElement so that it has to be calculated for every run.
      MUCH slower but works for XSpec and XSpec fails many tests without it.</xd:p>
    </xd:desc>
  </xd:doc>
  <xsl:variable name="resourceElement" select="/parent::*"/>

</xsl:stylesheet>
```