



SRX 2.0 Specification

OSCAR Recommendation, 7 April 2008

This version:

<http://www.lisa.org/fileadmin/standards/srx20.html>

Previous version:

<http://www.lisa.org/fileadmin/standards/srx10.html>

Editors:

David Pooley <dpooley@sdl.com>

Rodolfo M. Raya <rmraya@maxprograms.com>

Copyright © The Localization Industry Standards Association [LISA] 2006-2008. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to LISA.

The limited permissions granted above are perpetual and will not be revoked by LISA or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and LISA DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Abstract

This document defines the Segmentation Rules eXchange format (SRX). The purpose of the SRX format is to provide a standard method to describe segmentation rules that are being exchanged among tools and/or translation vendors.

Status of this document

This document is the current official recommendation from OSCAR for the SRX format. Comments may be sent to tmx_imp@lists.lisa-open.org.

Table of Contents

Abstract	2
Status of this document	2
1. Introduction	3
1.1. XML Compliance.....	3
1.2. Regular Expressions	3
1.2.1. Metacharacters	3
1.2.2. Operators	5
2. General structure	6
2.1. Language rules	6
2.2. Map rules.....	6
2.3. Example document	6
3. Detailed Specifications	7
3.1. Elements	7
3.2. Attributes	10
4. Implementation notes	13
5. Changes Since Previous Version (Non-Normative)	14
5.1. Backwards Compatibility	14
Appendices	15
A. References.....	15
B. Sample Document.....	16
C. Examples of Segmentation	18
D. XML Schema for SRX.....	19

1. Introduction

SRX defines an XML vocabulary for describing the rules used for breaking a text document into smaller fragments -or segments- suitable for translation.

SRX complements the TMX standard so that translation memory (TM) data that is exchanged between applications can be used more effectively. Having the segmentation rules that were used when a TM was created increases the leverage that can be achieved when deploying the TM data. SRX does not, however, address the following procedural issues, which would cause loss of leverage when TMX data is deployed in another environment:

- The use of segmentation rules that are different from those that were originally used to create the TM data
- TM data that was generated using a variety of segmentation rules

SRX is defined in two parts:

- A specification of the segmentation rules that are applicable for each language. This is represented by the `<language rules>` element.
- A specification of how the segmentation rules are applied to each language. This is represented by the `<map rules>` element.

1.1. XML Compliance

SRX is XML-compliant. SRX files are "well-formed" XML documents that can be processed without explicit reference to the SRX schema. However, a "valid" SRX file must conform to the SRX schema, and any suspicious SRX file should be verified against the SRX schema using a validating XML parser.

Since XML syntax is case sensitive, any XML application must define casing conventions. All elements and attributes names of SRX are defined in **lowercase**.

The SRX namespace is defined as "http://www.lisa.org/srx20". The following XML sample includes the SRX namespace definition.

```
<?xml version="1.0"?>
<myformat>
  <data>
    <srx xmlns="http://www.lisa.org/srx20" version="2.0">
      ... SRX data ...
    </srx>
  </data>
</myformat>
```

1.2. Regular Expressions

The segmentation rules themselves are represented using regular expressions. This allows for maximum flexibility in the definition of the rules. The following definitions are a subset of the current definition for the ICU regular expressions. Applications using other engines will need to adapt this format for use with their own parser.

1.2.1. Metacharacters

Character	Description
\a	Match a BELL, \u0007
\A	Match at the beginning of the input. Differs from ^ in that \A will not match after a new line within the input.
\b, outside of a [Set]	Match if the current position is a word boundary. Boundaries occur at the transitions between word (w) and non-word (W) characters, with combining marks ignored.
\b, within a [Set]	Match a BACKSPACE, \u0008.
\B	Match if the current position is not a word boundary.

<code>\cX</code>	Match a control-X character.
<code>\d</code>	Match any character with the Unicode General Category of Nd (Number, Decimal Digit.)
<code>\D</code>	Match any character that is not a decimal digit.
<code>\e</code>	Match an ESCAPE, <code>\u001B</code> .
<code>\E</code>	Terminates a <code>\Q ... \E</code> quoted sequence.
<code>\f</code>	Match a FORM FEED, <code>\u000C</code> .
<code>\G</code>	Match if the current position is at the end of the previous match.
<code>\n</code>	Match a LINE FEED, <code>\u000A</code> .
<code>\N{UNICODE CHARACTER NAME}</code>	Match the named character.
<code>\p{UNICODE PROPERTY NAME}</code>	Match any character with the specified Unicode Property.
<code>\P{UNICODE PROPERTY NAME}</code>	Match any character not having the specified Unicode Property.
<code>\Q</code>	Quotes all following characters until <code>\E</code> .
<code>\r</code>	Match a CARRIAGE RETURN, <code>\u000D</code> .
<code>\s</code>	Match a white space character. White space is defined as <code>[\t\n\r\f\rp{Z}]</code> .
<code>\S</code>	Match a non-white space character.
<code>\t</code>	Match a HORIZONTAL TABULATION, <code>\u0009</code> .
<code>\uhhhh</code>	Match the character with the hex value hhhh.
<code>\Uhhhhhhhh</code>	Match the character with the hex value hhhhhhhh. Exactly eight hex digits must be provided, even though the largest Unicode code point is <code>\U0010ffff</code> .
<code>\w</code>	Match a word character. Word characters are <code>[\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}]</code> .
<code>\W</code>	Match a non-word character.
<code>\x{hhhh}</code>	Match the character with hex value hhhh
<code>\xhh</code>	Match the character with two digit hex value hh
<code>\X</code>	Match a Grapheme Cluster
<code>\Z</code>	Match if the current position is at the end of input, but before the final line terminator, if one exists.
<code>\z</code>	Match if the current position is at the end of input.
<code>\Onnn</code>	Match the character with octal value nnn
<code>\n</code>	Back Reference. Match whatever the nth capturing group matched. n must be >1 and < total number of capture groups in the pattern
<code>[pattern]</code>	Match any one character from the set. See UnicodeSet for a full

	description of what may appear in the pattern.
.	Match any character.
^	Match at the beginning of a line.
\$	Match at the end of a line.
\	Quotes the following character. Characters that must be quoted to be treated as literals are * ? + [() { } ^ \$ \ . /

1.2.2. Operators

The following operators can be used.

Operator	Description
	Alternation. A B matches either A or B.
*	Match 0 or more times. Match as many times as possible.
+	Match 1 or more times. Match as many times as possible.
?	Match zero or one times. Prefer one.
{n}	Match exactly n times
{n,}	Match at least n times. Match as many times as possible.
{n,m}	Match between n and m times. Match as many times as possible, but not more than m.
*?	Match 0 or more times. Match as few times as possible.
+?	Match 1 or more times. Match as few times as possible.
??	Match zero or one times. Prefer zero.
{n}?	Match exactly n times
{n,}?	Match at least n times, but no more than required for an overall pattern match
{n,m}?	Match between n and m times. Match as few times as possible, but not less than n.
*+	Match 0 or more times. Match as many times as possible when first encountered, do not retry with fewer even if overall match fails (Possessive Match)
++	Match 1 or more times. Possessive match.
?+	Match zero or one times. Possessive match.
{n}+	Match exactly n times
{n,}+	Match at least n times. Possessive Match.
{n,m}+	Match between n and m times. Possessive Match.

2. General structure

An SRX document is enclosed in an `<srx>` root element. The `<srx>` element contains two elements: `<header>` and `<body>`. The `<header>` element contains zero, one, two or three `<formathandle>` elements followed by any number of elements from a foreign namespace. The `<body>` element contains two elements: `<languagerules>` and `<maprules>`.

2.1. Language rules

The `<languagerules>` element contains information about the segmentation rules for each particular language. It is a collection of `<languagerule>` elements. Each one of these contains a collection of `<rule>` elements.

Each `<rule>` element contains zero or one `<beforebreak>` element and zero or one `<afterbreak>` element which provide details of the regular expressions for the rules themselves. The `break` attribute indicates whether this is a segment break or an exception rule. The rules are applied in the order that they are specified within the `<languagerule>` element.

Note that this approach is an adaptation of the method described in [Unicode Technical Report 29](#) which covers text boundaries. Readers are encouraged to study this report with particular attention being given to the "sentence boundaries" section. Specifically, when a matching `<rule>` is found, the application should stop processing at that point and take the action as specified by the `break` attribute. As such, it is logical for the exceptions to be listed first in the `<languagerule>` element. The appropriate wording from TR29 is:

"Boundary determination is specified in terms of an ordered list of rules, indicating the status of a boundary position. The rules are numbered for reference and are applied in sequence to determine whether there is a boundary at a given offset. That is, there is an implicit "otherwise" at the front of each rule following the first. The rules are processed from top to bottom. As soon as a rule matches and produces a boundary status (boundary or no boundary) for that offset, the process is terminated."

2.2. Map rules

The `<maprules>` element contains information as to how each language should be segmented. It is a collection of `<langagemap>` elements that describe which rules should be used for each language.

2.2.1. Cascading

SRX supports the concept of "cascading" segmentation rules. The `cascade` attribute must be specified on the `<header>` element.

During segmentation processing, when a matching `<langagemap>` element is encountered, all the `<rule>` elements from the associated `<languagerule>` element are appended to an ordered list of `<rule>` elements. If the `cascade` attribute is set to "no" then aggregation of the `<rule>` elements will terminate. Once the ordered list of `<rule>` elements has been created, it is applied to the text to determine the segment boundaries.

2.3. Example document

See the [sample document](#) section for an example of a SRX document.

3. Detailed Specifications

3.1. Elements

This section lists the various elements used in the SRX document.

[<afterbreak>](#), [<beforebreak>](#), [<body>](#), [<formathandle>](#), [<header>](#), [<langagemap>](#), [<languagerule>](#), [<languagerules>](#), [<maprules>](#), [<rule>](#), [<srx>](#).

<afterbreak>

After break - The <afterbreak> element encloses a regular expression.

Required attributes:

None.

Optional attributes:

None.

Contents:

A regular expression which represents the text that appears after a segment break.

<beforebreak>

Before break - The <beforebreak> element encloses a regular expression.

Required attributes:

None.

Optional attributes:

None.

Contents:

A regular expression which represents the text that appears before a segment break.

<body>

Body - The <body> element encloses the language rules and language maps that are contained within the file.

Required attributes:

None.

Optional attributes:

None.

Contents:

One [<languagerules>](#) element and one [<maprules>](#) element.

<formathandle>

Format handling - The <formathandle> element determines how formatting that falls on a segment boundary

should be handled. The [type](#) attribute determines the type of formatting and the [include](#) attribute indicates how this formatting should be handled. As these elements are optional in the `<header>` element, the following defaults values apply:

```
<formathandle type="start"
              include="no"/> <formathandle type="end" include="yes"/>
<formathandle type="isolated" include="no"/>
```

See [Implementation notes](#) section for more details.

Required attributes:

[type](#), [include](#)

Optional attributes:

None

Contents:

None

<header>

Header - The `<header>` element contains information that is relevant to the whole document.

Required attributes:

[segmentsubflows](#), [cascade](#)

Optional attributes:

None

Contents:

Zero, one, two or three `<formathandle>` elements.
Followed by zero, one or more elements from foreign namespaces.

<languagemap>

Language map - The `<languagemap>` element maps one or more languages to a language rule.

Required attributes:

[languagepattern](#), [languagerulename](#)

Optional attributes:

None

Contents:

None.

<languagerule>

Language rule - The `<languagerule>` element encloses one instance of language rule data, a set of `<rule>` elements.

Required attributes:

[languagerulename](#)

Optional attributes:

None.

Contents:

One or more [<rule>](#) elements.

[<languageules>](#)

Language rules - The [<languageules>](#) element encloses the language rules data, the set of [<languageule>](#) elements.

Required attributes:

None.

Optional attributes:

None.

Contents:

One or more [<languageule>](#) elements.

[<maprules>](#)

Map rules - The [<maprules>](#) element encloses a set of [<languageumap>](#) elements. The order of the [<languageumap>](#) elements determines the logical order in which these rules should be applied.

Required attributes:

None.

Optional attributes:

None.

Contents:

One or more [<languageumap>](#) elements.

[<rule>](#)

Break or exception rule - The [<rule>](#) element defines a segmentation rule for a language using the [<beforebreak>](#) and [<afterbreak>](#) elements. The [break](#) attribute determines whether this is a rule that determines a break or an exception. If the [break](#) attribute is missing, it is assumed to be a break rule.

Required attributes:

None.

Optional attributes:

[break](#)

Contents:

Zero or one [<beforebreak>](#) element and zero or one [<afterbreak>](#) element.

The [<rule>](#) element must contain at least one child element.

<srx>

Root element - The <srx> element is the root element of the document. It encloses the header and body information for the file.

Required attributes:

[version](#)

Optional attributes:

None.

Contents:

One [<header>](#) element and one [<body>](#) element.

3.2. Attributes

This section lists the various attributes used in the SRX elements.

[break](#), [include](#), [languagepattern](#), [languagerulename](#), [segmentsubflows](#), [type](#), [version](#).

break

Break indicator - Specifies whether a rule is a break or an exception.

Value description:

A value of "no" indicates that the rule is an exception rule. A value of "yes" indicates that the rule is a break rule.

Default value:

"yes"

Used in:

[<rule>](#).

cascade

Cascade indicator - Specifies whether mapping rules should be cascaded.

Value description:

A value of "no" indicates that cascading should not be allowed. A value of "yes" indicates that rules should be cascaded. When cascading is not allowed, a matching [<languagepattern>](#) element terminates the search for a [<languagerule>](#) element. When cascading is allowed, all matching [<languagepattern>](#) elements are used to accumulate the rules from the associated [<languagerule>](#) elements.

Default value:

Undefined

Used in:

[<header>](#).

include

Formatting code behaviour - The include attribute indicates whether formatting is included in the segment being

created.

Value description:

A value of "no" indicates that the format code does not belong to the segment being created. A value of "yes" indicates that the format code belongs to the segment being created.

Default value:

Undefined

Used in:

[<formathandle>](#).

languagepattern

Language pattern - Identifies a language pattern.

Value description:

Specifies a regular expression for the language codes that map to the given language rule. Language codes are defined as in [\[RFC 4646\]](#).

Default value:

Undefined.

Used in:

[<languagemap>](#).

languagerulename

Language rule name - Specifies a unique name for a language rule.

Value description:

Used to link a language rule between the [<languagerule>](#) and [<languagemap>](#) elements.

Default value:

Undefined.

Used in:

[<languagerule>](#), [<languagemap>](#).

segmentsubflows

Subflow segmentation behaviour - The segmentsubflows attribute indicates how subflows should be segmented.

Value description:

A value of "no" indicates that subflows within a segment should not be segmented. A value of "yes" indicates that subflows should be segmented according to the rules. A subflow is defined as being a piece of text that appears within another segment but which should be handled separately. For example, in the following HTML snippet:

```
<p>Click ![Toolbar button. Click to preview.](../../../button.gif)
to preview the document.</p>
```

The text "Toolbar button. Click to preview." is a subflow. The segmentsubflows attribute determines whether this text should be segmented according to the rules.

Default value:

"yes"

Used in:

[<header>](#).

type

Formatting code type - The type attribute indicates the type of formatting for which the [<formathandle>](#) is being applied.

Value description:

This attribute can have one of three values. These are:

"start" to indicate the start of a pair of formatting codes

"end" to indicate the end of a pair of formatting codes

"isolated" to indicate a format that has no partner

Default value:

Undefined

Used in:

[<formathandle>](#).

version

SRX version - The version attribute indicates the version of the SRX format to which the document conforms.

Value description:

Fixed text: the major version number, a period, and the minor version number. For example: version="2.0".

Default value:

"2.0"

Used in:

[<srx>](#).

4. Implementation notes

This section has been included in order to clarify how the rules should be applied to the text which has been passed for segmentation.

The first task in the segmentation process is to build a list of rules to use. The language code of the source document is matched against the regular expressions contained in the `languagepattern` attribute of all `<languagemap>` elements. When there is a match, all rules of the corresponding `<languagerule>` element are added to the list of segmentation rules to use. If the `cascade` attribute is set to "no", the accumulation process stops after the first match. Otherwise, all `<languagemap>` elements must be examined.

Assuming that all applicable `<rule>` elements have already been accumulated into an ordered list, the algorithm described below illustrates exactly where segment boundaries should be detected. The following pseudo-code may not be the most efficient way to implement segmentation, but it fulfills the requirement of this specification.

```
for each inter-character position in the text
  for each <rule> element in the list
    if the current <rule> matches the inter-character position then
      if the <rule> element specifies a break then
        break the text at this point
      end if
    exit for
  end if
next
next
```

In a sentence such as "The U.K. Prime Minister, Mr. Blair, was seen out with his family today.", there are 70 possible inter-character breakpoints which need to be processed. By placing exceptions for "U.K." and "Mr." in `<rule break="no">` elements before the default **full stop followed by a space** `<rule break="yes">`, the necessary segmentation for this sentence can be achieved. See [Appendix C - Examples of Segmentation](#) for more details.

When the text being segmented contains inline markup, the following situations should be considered:

1. If a breaking condition happens right after an opening formatting mark, the value of the `include` attribute of the `<formathandle>` element with `type` attribute set to "start" should be examined and one of these actions should be taken:
 - If the value of the `include` attribute is "yes", the inline markup is included in the segment being closed.
 - If the value of the `include` attribute is "no" (default behaviour), the segment is terminated before the inline markup and the formatting code is incorporated to the following segment.
2. If a breaking condition happens right after a closing formatting mark, the value of the `include` attribute of the `<formathandle>` element with `type` attribute set to "end" should be examined and one of these actions should be taken:
 - If the value of the `include` attribute is "yes" (default behaviour), the inline markup is included in the segment being closed.
 - If the value of the `include` attribute is "no", the segment is terminated before the inline markup and the formatting code is incorporated to the following segment.
3. If a breaking condition happens right after an isolated formatting mark, the value of the `include` attribute of the `<formathandle>` element with `type` attribute set to "isolated" should be examined and one of these actions should be taken:
 - If the value of the `include` attribute is "yes", the inline markup is included in the segment being closed.
 - If the value of the `include` attribute is "no" (default behaviour), the segment is terminated before the inline markup and the formatting code is incorporated to the following segment.

5. Changes Since Previous Version (Non-Normative)

The main changes in this version (2.0) relative to the previous version (1.0) are as follows:

- Added [cascade](#) attribute to the `<header>` element.
- Removed `<maprule>` element and "maprulename" attribute.
- Required one `<languageules>` element and one `<maprules>` element as children of the `<body>` element.
- Defined the algorithm to use for segmenting text (see [Implementation notes](#) section).
- SRX 2.0 is based on XML Schema. This version does not include a DTD.
- Adopted [\[RFC 4646\]](#) for indicating language codes.
- Updated [normative references](#) to their latest versions.

5.1. Backwards Compatibility

SRX 2.0 is not compatible with previous version of SRX.

SRX 2.0 has introduced a new attribute which determines how language rules are aggregated (the [cascade](#) attribute on the `<header>` element). In previous versions of the standard, cascading behaviour was undefined which meant that implementations of the same SRX rules could vary between applications. Also please note that this version of the specification includes an [Implementation notes](#) section which clarifies how the rules should be applied to a block of text.

The element `<maprule>` has been removed to avoid potential differences in implementations, as SRX 1.0 didn't contain processing instructions for this element.

All existing implementations of SRX 1.0 will need to be adapted to support this new version of the specification.

A. References

[Unicode Character Database 5.0.0]

[Unicode Character Database 5.0.0](#) . Unicode Organisation, July 2006.

[RFC 4646]

[RFC 4646 Tags for Identifying Languages](#) . IETF (Internet Engineering Task Force), September 2006.

[XML 1.0]

[Extensible Markup Language \(XML\) 1.0 Second Edition](#) . W3C (World Wide Web Consortium), September 2006.

[ICU Regular Expressions]

[ICU Regular Expressions User Guide](#) . IBM and Others, December 2006.

[Unicode Technical Report 29]

[Unicode Standard Annex #29: Text Boundaries](#) , Unicode Consortium, October 2006.

B. Sample Document

In this example of an SRX document indentations are added for ease of reading, and the different types of notation are mixed to illustrate the various possibilities.

```
<?xml version="1.0"?>
<srx version="2.0"
  xmlns="http://www.lisa.org/srx20"
  xsi:schemaLocation="http://www.lisa.org/srx20 srx20.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <header segmentsubflows="yes" cascade="yes">
    <formathandle type="start" include="no"/>
    <formathandle type="end" include="yes"/>
    <formathandle type="isolated" include="yes"/>
  </header>
  <body>
    <languagegerules>
      <languagegerule languagegerulename="Default">
        <!-- Common rules for most languages -->
        <rule break="no">
          <beforebreak>^\s*[0-9]+\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
        <rule break="yes">
          <afterbreak>\n</afterbreak>
        </rule>
        <rule break="yes">
          <beforebreak>[\.\?!]+\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
      </languagegerule>
      <languagegerule languagegerulename="English">
        <!-- Some English abbreviations -->
        <rule break="no">
          <beforebreak>\s[Ee][Tt][Cc]\./</beforebreak>
          <afterbreak>\s[a-z]</afterbreak>
        </rule>
        <rule break="no">
          <beforebreak>\sMr\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
        <rule break="no">
          <beforebreak>\sU.K\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
      </languagegerule>
      <languagegerule languagegerulename="French">
        <!-- Some French abbreviations -->
        <rule break="no">
          <beforebreak>\s[Mm]lle\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
        <rule break="no">
          <beforebreak>\s[Mm]lles\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
        <rule break="no">
          <beforebreak>\s[Mm]me\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
        <rule break="no">
          <beforebreak>\s[Mm]mes\./</beforebreak>
          <afterbreak>\s</afterbreak>
        </rule>
      </languagegerule>
      <languagegerule languagegerulename="Japanese">
        <!-- Rules for breaking on Japanese punctuation
          \xff61: Halfwidth ideographic full stop
          \x3002: Ideographic full stop
          \xff0e: Fullwidth full stop
          \xff1f: Fullwidth question mark
          \xff01: Fullwidth exclamation mark
```

```
-->
<rule break="yes">
  <beforebreak>[\xff61\x3002\xff0e\xff1f\xff01]+</beforebreak>
  <afterbreak></afterbreak>
</rule>
</languagerule>
</languagerules>
<maprules>
  <!-- List exceptions first -->
  <languagemap languagepattern="[Ee][Nn].*" languagerulename="English"/>
  <languagemap languagepattern="[Ff][Rr].*" languagerulename="French"/>
  <!-- Japanese breaking rules -->
  <languagemap languagepattern="[Jj][Aa].*" languagerulename="Japanese"/>
  <!-- Common breaking rules -->
  <languagemap languagepattern=".*" languagerulename="Default"/>
</maprules>
</body>
</srx>
```

C. Examples of Segmentation

This section provides some examples of how segmentation rules might be applied to fragments of text. These are simple examples and are by no means a complete reference to segmentation.

Rule set	Text to segment	Result	Notes
<pre><rule break="yes"> <beforebreak>[\.\?!]+</beforebreak> <afterbreak>\s</afterbreak> </rule></pre>	The U.K. Prime Minister, Mr. Blair, was seen out with his family today.	(1) The U.K. (2) Prime Minister, Mr. (3) Blair, was seen out with his family today	The simple full-stop followed by a space rule here showing its limitations
<pre><rule break="no"> <beforebreak>\sU\.K\.</beforebreak> <afterbreak>\s</afterbreak> </rule> <rule break="yes"> <beforebreak>[\.\?!]+</beforebreak> <afterbreak>\s</afterbreak> </rule></pre>	The U.K. Prime Minister, Mr. Blair, was seen out with his family today.	(1) The U.K. Prime Minister, Mr. (2) Blair, was seen out with his family today	Partially corrected with an exception for "U.K."
<pre><rule break="no"> <beforebreak>\sU\.K\.</beforebreak> <afterbreak>\s</afterbreak> </rule> <rule break="no"> <beforebreak>Mr.</beforebreak> <afterbreak>\s</afterbreak> </rule> <rule break="yes"> <beforebreak>[\.\?!]+</beforebreak> <afterbreak>\s</afterbreak> </rule></pre>	The U.K. Prime Minister, Mr. Blair, was seen out with his family today.	(1) The U.K. Prime Minister, Mr. Blair, was seen out with his family today	Sufficient exceptions to prevent segmentation on "U.K." and "Mr."

Added <header> and <body> elements.

Nov-22-2002 by DRP: First draft version

-->

```
<xs:schema xmlns:srx="http://www.lisa.org/srx20"
  targetNamespace="http://www.lisa.org/srx20" xml:lang="en"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:element name="afterbreak">
    <xs:annotation>
      <xs:documentation>Contains the regular expression to match before the
segment
      break</xs:documentation>
    </xs:annotation>
    <xs:complexType mixed="true"/>
  </xs:element>
  <xs:element name="beforebreak">
    <xs:annotation>
      <xs:documentation>Contains the regular expression to match after the
segment
      break</xs:documentation>
    </xs:annotation>
    <xs:complexType mixed="true"/>
  </xs:element>
  <xs:element name="body">
    <xs:annotation>
      <xs:documentation>SRX body</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="srx:languageRules"/>
        <xs:element ref="srx:maprules"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="formatHandle">
    <xs:annotation>
      <xs:documentation>Determines which side of the segment break that
formatting
      information goes</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:attribute name="include" use="required">
        <xs:annotation>
          <xs:documentation>A value of "no" indicates that the format code
does not belong
          to the segment being created. A value of "yes" indicates that
the format code
          belongs to the segment being created.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="yes"/>
            <xs:enumeration value="no"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="type" use="required">
        <xs:annotation>
          <xs:documentation>The type of format for which behaviour is being
defined. Can be
          "start", "end" or "isolated".</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="start"/>
            <xs:enumeration value="end"/>
            <xs:enumeration value="isolated"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="header">
```

```

<xs:annotation>
  <xs:documentation>SRX header</xs:documentation>
</xs:annotation>
<xs:complexType>
  <xs:sequence>
    <xs:element ref="srx:formathandle" minOccurs="0" maxOccurs="3"/>
    <xs:any maxOccurs="unbounded" namespace="##other"
processContents="lax"/>
  </xs:sequence>
  <xs:attribute name="segmentsubflows" use="required">
    <xs:annotation>
      <xs:documentation>Determines whether text subflows should be
segmented</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="yes"/>
        <xs:enumeration value="no"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="cascade" use="required">
    <xs:annotation>
      <xs:documentation>Determines whether a matching
&lt;languagemap&gt; element
should terminate the search</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="yes"/>
        <xs:enumeration value="no"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="languagemap">
  <xs:annotation>
    <xs:documentation>Maps one or more languages to a set of
rules</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="languagerulename" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>The name of the language rule to use when the
languagepattern
regular expression is satisfied</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="languagepattern" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>The regular expression pattern match for the
language
code</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="languagerule">
  <xs:annotation>
    <xs:documentation>A set of rules for a logical set of
languages</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="srx:rule" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="languagerulename" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>The name of the language
rule</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="languagerules">
  <xs:annotation>
    <xs:documentation>Contains all the logical sets of
rules</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="srx:languagerule" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="maprules">
  <xs:annotation>
    <xs:documentation>A set of language maps</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="srx:langagemap" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="rule">
  <xs:annotation>
    <xs:documentation>A break/no break rule</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="srx:beforebreak" minOccurs="0"/>
      <xs:element ref="srx:afterbreak" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="break">
      <xs:annotation>
        <xs:documentation>Determines whether this is a segment break or
an exception
          rule</xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="yes"/>
          <xs:enumeration value="no"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="srx">
  <xs:annotation>
    <xs:documentation>OSCAR Segmentation Rules eXchange</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="srx:header"/>
      <xs:element ref="srx:body"/>
    </xs:sequence>
    <xs:attribute name="version" use="required">
      <xs:annotation>
        <xs:documentation>The version of SRX</xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="2.0"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
</xs:schema>

```