Expressing IDNA2008 Context and Bidi Rules in the XML Format for Representing Label Generation Rulesets

This document presents the IDNA 2008 Context Rules as defined in RFC5892 Appendix A. "Contextual Rules Registry" and their equivalents from "Representing Label Generation Rulesets in XML". Appendix B presents the equivalent of the “Bidi Rule” from RFC5893 “Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)” for a single label.

Comparison of Definitions and Syntax Elements

This section presents the various definitions and syntax element from Appendix A of RFC5892 and compares them with syntax elements available in the XML Format. Text excerpted from RFC5892 is shown indented and in blue with emphasis added.

> **cp** represents the code point to be tested.

To match a specific code point the XML format uses a literal <char cp="xxxx" /> where xxxx is a single code point (but may also be a sequence). In some types of rule, the <anchor /> element is used to mark the location of the code point to be tested.

> **FirstChar** is a special term that denotes the first code point in a label.
> **LastChar** is a special term that denotes the last code point in a label.
> **.is.** represents checking the position in a label.
> **A .is. B** evaluates to True if A and B have same position in the same label.

The corresponding construct in the XML format are <start/> and <end/>, which denote the position before the first or past the last code point. The **.is.** operator is typically used to express **cp .is. FirstChar**, which would be represented as <start /><char cp="xxxx"/> in the XML format.

> **.eq.** represents the equality relation.
> **A .eq. B** evaluates to True if A equals B.
> **.ne.** represents the non-equality relation.
> **A .ne. B** evaluates to True if A is not equal to B.

The XML format does not have direct equivalents to these.

> **.in.** represents the set inclusion relation.
> **A .in. B** evaluates to True if A is a member of the set B.

In the XML format, sets are represented as character classes. Predefined classes can be referenced by name for example as <class by-ref="name"/>.

A functional notation, Function_Name(cp), is used to express either string positions within a label, Boolean character property tests of a code point, or a regular expression match. When such function names refer to Boolean character property tests, the function names use the exact Unicode character property name for the property in question, and "cp" is evaluated as the Unicode value of the code point to be tested, rather than as its position in the label. When such function names refer to string positions within a label, "cp" is evaluated as its position in the label.

The XML format has no feature that directly corresponds to this functional notation, but equivalent results can be specified by slightly different means. The XML format is able to express all the rules given in Appendix A.

RegExpMatch(X) takes as its parameter X a schematic regular expression consisting of a mix of Unicode character property values and literal Unicode code points.

In the XML format a <rule> represents a regular expression match with a limited subset of regular expression features required for IDN context rules. <rule> elements can be nested. The following match operators are defined.

Type	Operator	Examples
logical	any	<any />

	choice	<choice>
		  <rule by-ref="alternative1"/>
		  <rule by-ref="alternative2"/>
		</choice>

positional	start	<start />
	end	<end />
literal	char	<char cp="0061 0062 0063" />
set	class	<class by-ref="class1" /> <class>0061 0064-0065</class>
nested	rule	<rule by-ref="rule1" /> <rule><any /><rule />
contextual	anchor	<anchor />
	look-ahead	<look-ahead><any /></look-ahead>
	look-behind	<look-behind><any /></look-behind>

Other than positional and contextual operators, these allow a count attribute to specify both fixed as well as minimal repetition counts. With them, and using an anonymous <rule> element like a "group" operator, a significant subset of common regular expression functionality can be expressed. The expectation Is that this subset is more than sufficient to cater to the needs of Label Generation Rulesets including any future rules for IDNA2008.

Script(cp) returns the value of the Unicode Script property, as defined in Scripts.txt in the Unicode Character Database.

Canonical_Combining_Class(cp) returns the value of the Unicode Canonical_Combining_Class property, as defined in UnicodeData.txt in the Unicode Character Database.

The XML format has a more generic feature, that allows to create a set (<class>) from any Unicode property. That set can then be tested for membership. <class property="sc:Grek" /> would be equivalent to Script(cp) =Grek, and <class name="virama" property="ccc:9" /> would match all cp's with a canonical combining class value of 9 (virama characters).

Before(cp) returns the code point of the character immediately preceding cp in logical order in the string representing the label. **Before(FirstChar)** evaluates to Undefined.

After(cp) returns the code point of the character immediately following cp in logical order in the string representing the label. **After(LastChar)** evaluates to Undefined.

Note that "Before" and "After" do not refer to the visual display order of the character in a label, which may be reversed or otherwise modified by the bidirectional algorithm for labels including characters from scripts written right to left. Instead, "Before" and "After" refer to the network order of the character in the label.

The XML format provides <look-behind> and <look-ahead> operators which represent regular expressions that must be satisfied immediately before or after a tested cp (represented as <anchor />).

The clauses "Then True" and "Then False" imply exit from the pseudo-code routine with the corresponding result.

In the XML format an <action> element defines whether a <rule> should match or not match, and defines the resulting policy state (which can go beyond valid/invalid) for the label. Likewise a context rule (when rule) is leads to a state of invalid if the rule is not matched by the label.

Repeated evaluation for all characters in a label makes use of the special construct:

```
For All Characters:
    Expression;
End For;
```

This construct requires repeated evaluation of "Expression" for each code point in the label, starting from FirstChar and proceeding to LastChar.

The different fields in the rules are to be interpreted as follows:

Code point:

The code point, or code points, to which this rule is to be applied. Normally, this implies that if any of the code points in a label is as defined, then the rules should be applied. If evaluated to True, the code point is OK as used; if evaluated to False, it is not OK.

Validating a label against an LGR Ruleset makes use of a small number of Implicit Actions which include such looping over all characters.

In the XML format, some rules are triggered by <action> elements, but some are triggered by "when" and "not-when" attributes associated with code points and variant mappings. Whenever a label contains a code point with a "when" rule, that rule must be matched, or the code point is considered invalid. The actual code point can be generically referred to in the rule with a <anchor /> element.

"Not-when" rules must not be matched unless a code point is invalid.

Variant mappings can have similar "when" and "not-when" rules. Instead of defining validity of the code point, the rules for variant mappings define whether a mapping exists or not. A mapping that does not exist in a given context is ignored in processing.

Overview:
A description of the goal with the rule, in plain English.

Lookup:
True if application of this rule is recommended at lookup time;
False otherwise.

Rule Set:
The rule set itself, as described above.

Other features

Specifying character sets for matching. In addition to being based on Unicode properties, character classes can be defined by explicit list (in long and short notation) and by reference to "tag" values associated with code points (this combines the functions of the code point table with that of a set definition table).

Classes can be combined and modified with the usual set operators (union, intersection, negation, difference and symmetric difference).

A <rule> when nested inside another <rule> is anonymous and acts much like a group operator in a regular expression. A <choice/> element allows multiple options, just like a regex "or" operator.

A special token <any /> acts like the wildcard operator.

All operators, except those that by their nature are singletons, accept a count, or minimal count. The standard ".*" of the regular expression syntax would be rendered as <any count="0+" />

With that we are ready to turn to the actual rules from IDNA 2008 and see how they get expressed in this notation.

Appendix A.1. ZERO WIDTH NON-JOINER

```
Code point:
   U+200C

Overview:
   This may occur in a formally cursive script (such as Arabic) in a
   context where it breaks a cursive connection as required for
   orthographic rules, as in the Persian language, for example.  It
   also may occur in Indic scripts in a consonant-conjunct context
   (immediately following a virama), to control required display of
   such conjuncts.

Lookup:
   True

Rule Set:

   False;

   If Canonical_Combining_Class(Before(cp)) .eq.  Virama Then True;

   If RegExpMatch((Joining_Type:{L,D})(Joining_Type:T)*\u200C

      (Joining_Type:T)*(Joining_Type:{R,D})) Then True;
```

```
<char cp="200C" when="non-joiner" />
...
<class name="left-joining" property="jt:L" />
<class name="dual-joining" property="jt:D" />
<class name="right-joining" property="jt:R" />
<class name="transparent" property="jt:T" />
<rule name="non-joiner" >
   <choice>
      <!-- references rule from A.2 -->
      <rule by-ref="joiner" />
      <rule>
         <look-behind>
            <union>
               <class by-ref="left-joining" />
               <class by-ref="dual-joining" />
            </union>
            <class by-ref="transparent" count="0+" />
         </look-behind>
         <anchor />
         <look-ahead>
            <class by-ref="transparent" count="0+" />
            <union>
               <class by-ref="right-joining" />
               <class by-ref="dual-joining" />
            </union>
         <look-ahead/>
      </rule>
   </choice>
</rule>
```

Appendix A.2. ZERO WIDTH JOINER

```
   Code point:
      U+200D

   Overview:
      This may occur in Indic scripts in a consonant-conjunct context
      (immediately following a virama), to control required display of
      such conjuncts.

   Lookup:
      True

   Rule Set:

      False;

      If Canonical_Combining_Class(Before(cp)) .eq.  Virama Then True;
```

```
    <char cp="200D" when="joiner" />
    ...
    <class name="virama" property="ccc:9" />
    <rule name="joiner">
       <look-behind>
          <class by-ref="virama" />
       </look-behind>
       <anchor />
   </rule>
```

Appendix A.3. MIDDLE DOT

```
   Code point:
      U+00B7

   Overview:
      Between 'l' (U+006C) characters only, used to permit the Catalan
      character ela geminada to be expressed.

   Lookup:
      False

   Rule Set:

      False;

      If Before(cp) .eq.  U+006C And

         After(cp) .eq.  U+006C Then True;
```

```
        <char cp="00B7" when="catalan-middle-dot" />
        ...
        <rule name="catalan-middle-dot">
                <look-behind>
                        <char cp="006C" />
                </look-behind>
                <anchor />
                <look-ahead>
                        <char cp="006C" />
                </look-ahead>
        </rule>
```

Appendix A.4. GREEK LOWER NUMERAL SIGN (KERAIA)

```
   Code point:
      U+0375

   Overview:
      The script of the following character MUST be Greek.

   Lookup:
      False

   Rule Set:

      False;

      If Script(After(cp)) .eq.  Greek Then True;
```

```
    <char cp="0375" when="precedes-greek" />
    ...
    <rule name="precedes-greek">
      <anchor />
      <look-ahead>
        <class property="sc:Grek" />
      </look-ahead>
    </rule>
```

Appendix A.5. HEBREW PUNCTUATION GERESH

```
   Code point:
      U+05F3

   Overview:
      The script of the preceding character MUST be Hebrew.

   Lookup:
      False

   Rule Set:
      False;

      If Script(Before(cp)) .eq.  Hebrew Then True;
```

```
    <char cp="05F3" when="follows-hebrew">
    ...
    <rule name="follows-hebrew" >
      <look-behind>
        <class property="sc:Hebrew" />
      </look-behind>
      <anchor />
    </rule>
```

Appendix A.6.   HEBREW PUNCTUATION GERSHAYIM

```
   Code point:
      U+05F4

   Overview:
      The script of the preceding character MUST be Hebrew.

   Lookup:
      False

   Rule Set:

      False;

      If Script(Before(cp)) .eq.  Hebrew Then True;
```

```
    <char cp="05F4" when="follows-hebrew">
    ...
    <!-- rule can be reused from A.5 -->
    <rule name="follows-hebrew" >
      <look-behind>
        <class property="sc:Hebr" />
      </look-behind>
      <anchor />
    </rule>
```

Appendix A.7.   KATAKANA MIDDLE DOT

```
   Code point:
      U+30FB

   Overview:
      Note that the Script of Katakana Middle Dot is not any of
      "Hiragana", "Katakana", or "Han".  The effect of this rule is to
      require at least one character in the label to be in one of those
```

scripts.

Lookup:
   False

Rule Set:

```
   False;

   For All Characters:

      If Script(cp) .in. {Hiragana, Katakana, Han} Then True;

   End For;
```

```
      <char cp="300B" when="japanese-in-label">
      ...
      <rule name="japanese-in-label" >
         <union>
              <class property="sc:Hani"/>
              <class property="sc:Kata"/>
              <class property="sc:Hira"/>
          </union>
      </rule>
```

Appendix A.8. ARABIC-INDIC DIGITS

Code point:
   0660..0669

Overview:
   Can not be mixed with Extended Arabic-Indic Digits.

Lookup:
   False

Rule Set:

```
   True;

   For All Characters:

      If cp .in. 06F0..06F9 Then False;

   End For;
```

```
    <range first-cp="0660" last-cp="0669" not-when="extended-arabic-digits"/>
    ...
    <rule name="extended-arabic-digits">
       <class>06F0-06F9</class>
    </rule>
```

Appendix A.9. EXTENDED ARABIC-INDIC DIGITS

```
Code point:
   06F0..06F9

Overview:
   Can not be mixed with Arabic-Indic Digits.

Lookup:
   False

Rule Set:

   True;

   For All Characters:

      If cp .in. 0660..0669 Then False;

   End For;
```

```
 <range first-cp="06F0" last-cp="06F9" not-when="arabic-indic-digits"/>
 ...
 <rule name="arabic-indic-digits">
    <class>0660-0669</class>
 </rule>
```

Appendix B.1 RFC 5893 “Bidi Rule”

Implementation of the “Bidi Rule” from Section 2 of RFC 5893 “Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)”.

Defining these common collection of bidi properties,

```
<union name="numeric-European">
   <class property="bc:EN" />
   <class property="bc:ET" />
   <class property="bc:ES" />
   <class property="bc:CS" />
</union>
<union  name="neutrals" >
   <class property="bc:ON" />
   <class property="bc:BN" />
</union>
```

the conditions that make up the “Bidi Rule” can be implemented as follows:

Left-to-Right (LTR) Label

1. The first character must be a character with Bidi property L, R, or AL. If it has the R or AL property, it is an RTL label; if it has the L property, it is an LTR label.

5. In an LTR label, only characters with the Bidi properties L, EN, ES, CS, ET, ON, BN, or NSM are allowed.

6. In an LTR label, the end of the label must be a character with Bidi property L or EN, followed by zero or more characters with Bidi property NSM.

```
<union name="LTR-bidi-classes">
  <class property="bc:L" />
  <class property="bc:NSM" />
  <class by-ref="numeric-European" />
  <class by-ref="neutrals" />
</union>

<rule name="LTR-label" comment=”conditions 1, 5, and 6”>
  <start />
  <class property="bc:L" />
  <class property="bc:NSM" count="0+" />
  <class by-ref="LTR-bidi-classes" count="0+" />
  <choice>
    <class property="bc:L" />
    <class property="bc:EN" />
  </choice>
  <class property="bc:NSM" count="0+" />
  <end />
</rule>
```

Right-To-Left (RTL) Label

1. The first character must be a character with Bidi property L, R, or AL. If it has the R or AL property, it is an RTL label; if it has the L property, it is an LTR label.

2. In an RTL label, only characters with the Bidi properties R, AL, AN, EN, ES, CS, ET, ON, BN, or NSM are allowed.

3. In an RTL label, the end of the label must be a character with Bidi property R, AL, EN, or AN, followed by zero or more characters with Bidi property NSM.

```
<union name="RTL-bidi-classes">
  <class property="bc:R"/>
  <class property="bc:AL"/>
  <class property="bc:AN"/>
  <class by-ref="numeric-European" />
  <class by-ref="neutrals" />
</union>

<rule name="RTL-label" comment=”conditions 1, 2, and 3”>
  <start />
  <choice>
    <class property="bc:R" />
    <class property="bc:AL" />
  </choice>
  <class by-ref="LTR-bidi-classes" count="0+" />
  <choice>
    <class property="bc:R" />
    <class property="bc:AL" />
    <class property="bc:AN" />
    <class property="bc:EN" />
  </choice>
  <class property="bc:NSM" count="0+" />
  <end />
</rule>
<rule name="bidi-rule" comment=”condition 1 and 1,2,3,5,6 by reference”>
  <choice>
    <rule by-ref="LTR-label" />
    <rule by-ref="RTL-label" />
  </choice>
</rule>
<action disp="invalid" not-match="bidi-rule" />
```

Prohibition on Mixed Numbers

4. In an RTL label, if an EN is present, no AN may be present, and vice versa.

```
<rule name="mixed-digits" comment=”condition 4”>
  <choice>
    <rule>
      <class property="bc:EN" />
      <any count="0+" />
      <class property="bc:AN" />
    </rule>
    <rule>
      <class property="bc:AN" />
      <any count="0+" />
      <class property="bc:EN" />
    </rule>
  </choice>
</rule>

<action disp="invalid" match="mixed-digits" />
```