Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 27, 2014

K. Davies
ICANN
A. Freytag
ASMUS Inc.

September 23, 2013

Representing Label Generation Rulesets using XML
draft-davies-idntables-04

Abstract

This memo describes a method of representing the domain name registration policy for a zone administrator using Extensible Markup Language (XML). These policies, known as "Label Generation Rulesets" (LGRs), are particularly used for the implementation of Internationalised Domain Names (IDNs). The rulesets are used to implement and share policy on which specific Unicode code points are permitted for registrations, which alternative code points are considered variants, and what actions may be performed on those variants.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 27, 2014.

Table of Contents

1. Introduction

This memo describes a method of using Extensible Markup Language (XML) to describe the algorithm used to determine whether a given domain label is permitted, and under which conditions based on the code points it contains and their context. These algorithms are comprised of a list of permissible code points, variants, and a number of rules describing where certain relationships are applied. These algorithms form part of a zone administrator's policies, and can be referred to as Label Generation Rulesets (LGRs), or IDN tables.

Administrators of the zones for top-level domain registries have historically published their LGRs using ASCII text or HTML. The formatting of these documents has been loosely based on the format used for the Language Variant Table in [RFC3743]. [RFC4290] also provides a "model table format" that describes a similar set of functionality.

Through the first decade of IDN deployment, experience has shown that LGRs derived from these formats are difficult to consistently implement and compare due to their differing formats. A universal format, such as one using a structured XML format, will assist by improving machine-readability, consistency, reusability and maintainability of LGRs. It also provides for more complex conditional implementation of variants that reflects the known requirements of current zone administrator policies.

While the predominant usage of this specification is to represent IDN label policy, the format is not limited to IDN usage may also be used for describing ASCII domain name label rulesets.

2. Design Goals

The following items are explicit design goals of this format:

- MUST be in a format that can be implemented in a reasonably straightforward manner in software;
- The format SHOULD be able to be checked for formatting errors, such that common mistakes can be caught;
- An LGR MUST be able to express the set of valid code points that are allowed for registration under a specific zone administrator's policies;
- MUST be able to express computed alternatives to a given domain name based on a one-to-one, or one-to-many relationship. These computed alternatives are commonly known as "variants";
- Variants SHOULD be able to be tagged with specific categories, such that the categories can be used to support registry policy (such as whether to list the computed variant in the zone, or to merely block it from registration);
- Variants MUST be able to stipulated based on contextual information. For example, specific variants may only be applicable when they follow another specific code point, or when the code point is displayed in a specific presentation form;
- The data contained within an LGR MUST be unambiguous, such that independent implementations that utilise the contents will arrive at the same results;
- LGRs SHOULD be suitable for comparison and re-use, such that one could easily compare the contents of two or more to see the differences, to merge them, and so on.
- As many existing IDN tables are practicable SHOULD be able to be migrated to the LGR format with all applicable logic retained.

It is explicitly NOT the goal of this format to stipulate what code points should be listed in an LGR by a zone administrator. Which registration policies are used for a particular zone is outside the scope of this memo.

3. Requirements

To be able to fulfil the known utilisation of LGRs, the existing corpus of published IDN tables were reviewed to prepare this specification.

In addition, the requirements of ICANN's work to implement an LGR for the DNS Root Zone [LGR-PROCEDURE] were also considered. In Section B of that document, five specific requirements for an LGR methodology were identified:

- The ability to identify a set of code points that are permitted.
- The ability to represent a list of variants, if any, for each code point.
- A method of identifying code points that are related, using a tag.
- The ability to describe rules regarding the possible actions that may be performed on the resulting label (such as blocked, allocatable, etc.)
- The ability to describe rules that check for ill-formed combinations across the whole label.

Finally, the syntax and rules in [RFC5892] were reviewed.

4. LGR Format

An LGR is expressed as a well-formed XML Document[XML].

4.1. Namespace

The XML Namespace URI is [TBD].

4.2. Basic structure

The basic XML framework of the document is as follows:

```
<?xml version="1.0"?>
<lgr xmlns="http://www.iana.org/lgr/0.1">
    ...
</lgr>
```

Within the "lgr" element rests several sub-elements. Firstly is a "meta" element that contains all meta-data associated with the IDN table, such as its authorship, what it is used for, implementation notes and references. This is followed by a "data" element that contains the substantive code point data. Finally, an optional "rules" element contains information on whole-label evaluation rules, if any, along with any specific rules regarding the disposition of computed variants.

```
<?xml version="1.0"?>
<lgr xmlns="http://www.iana.org/lgr/0.1">
    <meta>
        ...
    </meta>
    <data>
        ...
    </data>
    <rules>
        ...
    </rules>
</lgr>
```

A document MUST contain exactly one "lgr" element. Each "lgr" element MUST contain exactly one "data" element, optionally preceded by one "meta" element and optionally followed by one "rules" element.

4.3. Metadata

The "meta" element is used to express meta-data associated within the LGR. It can be used to identify the author or relevant contact person, explain the intended usage of the LGR, and provide

implementation notes as well as references. The data contained within is not required by software consuming the LGR in order to calculate valid labels, or to calculate variants. However, the "unicode-version" element should be used to identify whether a consumer of the table has the right Unicode data to perform operations on the table.

4.3.1. The version element

The "version" element is used to uniquely identify each version of the LGR being represented. No specific format is required, but it is RECOMMENDED that it be a numerical positive integer, which is incremented with each revision of the file.

An example of a typical first edition of a document:

```
<version>1</version>
```

4.3.2. The date element

The "date" element is used to identify the date the LGR was posted. The contents of this element MUST be a valid ISO 8601 date string as described in[RFC3339].

Example of a date:

```
<date>2009-11-01</date>
```

4.3.3. The language element

The "language" element signals that the LGR is associated with a specific language or script. The value of the language element must be a valid language tag as described in[RFC5646]. The tag may simply refer to a script if the LGR is not referring to a specific language.

Example of an English language LGR:

```
<language>en</language>
```

If the LGR applies to a specific script, rather than a language, the "und" language tag should be used followed by the relevant [RFC5646] script subtag. For example, for a Cyrillic script LGR:

```
<language>und-Cyrl</language>
```

If the LGR covers a specific set of multiple languages or scripts, the language element can be repeated. However, for cases of

insignificant admixture of characters from other scripts, the use of a single "language" element identifying the predominant script is RECOMMENDED. In the exceptional case where no script is predominant, use Zyyy (Common):

```
<language>und-Zyyy</language>
```

Note that that for the particular case of Japanese, a script tag "Japn" exists that matches the mixture of scripts used in writing that language. The preferred language element would be:

```
<language>und-Japn</language>
```

4.3.4. The domain element

This optional element refers to a domain to which this policy is applied.

```
<domain>example.com</domain>
```

There may be multiple <domain> tags used to reflect a list of domains.

4.3.5. The description element

The "description" element is a free-form element that contains any additional relevant description that is useful for the user in its interpretation. Typically, this field contains authorship information, as well as additional context on how the LGR was formulated (such as citations and references), and how it has been applied.

The element has an optional "type" attribute, which refers to the media type of the enclosed data. If the description lacks a type field, it will be assumed to be plain text.

The "type" attribute may be used to specify the encoding within description element. The attribute should be a valid MIME type. If supplied, it will be assumed the contents is content of that encoding. Typical types would be "text/plain" or "text/html". "text/plain" will be assumed if no type attribute is specified.

4.3.6. The validity-start and validity-end elements

The "validity-start" and "validity-end" elements are optional elements that describe the time period from which the contents of the LGR become valid (i.e. are used in registry policy), and the contents of the LGR cease to be used.

The times should conform to the format described in section 5.6 of [RFC5646]. It may be comprised of a date, or a date and time stamp.

4.3.7. The unicode-version element

Whenever an IDN table depends on character properties from a given version of the Unicode standard, the minimum version number MUST be listed. If any software processing the table does not have the minimum requisite version, it MUST NOT perform any operations relating to whole-label evaluation. This is because some Unicode code points may not have been assigned in an earlier version, leaving properties for these code points undefined. It is RECOMMENDED to only reference stable or immutable properties as other may change between versions.

```
<unicode-version>6.2</unicode-version>
```

It is not necessary to include a "unicode-version" element for files that do not make use of Unicode properties. Because Unicode has been strictly additive from Version 1.1, the required minimum version for the repertoire can be uniquely determined by checking the code point values in any "cp" attributes against the "age" property in [UAX42].

4.3.8. The references element

An IDN table may define a list of references in an optional "references" element. The references element contains any number of "reference" elements, each containing an "id" attribute. It is RECOMMENDED that the "id" attribute be an integer. The "id" attributes MUST be unique. The value of the element may a citation of a standard, dictionary or other specification, in any suitable format. A reference can be associated with many elements contained in the "data" or "rules" elements, by using an optional "ref" attribute.

```
<references>
  <reference id="0">The Unicode Standard, Version 7.0</reference>
  <reference id="1">Big-5: Computer Chinese Glyph and Character
     Code Mapping Table, Technical Report C-26, 1984</reference>
  <reference id="2">ISO/IEC 10646:2012 3rd edition</reference>
  ...
</references>
...
<data>
  <char cp="0620" ref="0" />
  ...
</data>
```

A "ref" attribute may not occur on elements that are named references to character classes and rules and on certain specific other element types. See description of these elements below.

5. Code Point Rules

The bulk of a label generation ruleset is a description of which set of code points are eligible for a given label. For rulesets that perform operations that result in potential variants, the code point-level relationships between variants need to also be described.

The code point data is collected within a "data" element. Within this element, a series of "char" and "range" elements describe eligible code points, or ranges of code points, respectively.

Discrete permissible code points or code point sequences are declared with a "char" element, e.g.

```
<char cp="002D"/>
```

Ranges of permissible code points may be stipulated with a "range" element, e.g.

```
<range first-cp="0030" last-cp="0039"/>
```

The range is inclusive of the first and last code points. Whether code points are specified individually or as part of a range makes no difference in processing the data, and tools reading or writing the XML format may not retain a distinction. All attributes defined for a range element are as if applied to each code point within.

Code points must be expressed in uppercase, hexadecimal, and zero padded to a minimum of 4 digits - in other words according to the standard Unicode convention but without the prefix "U+". The rationale for not allowing other encoding formats, including native Unicode encoding in XML, is explored in[UAX42]. The XML conventions used in this format, including the element and attribute names, mirror this document where practical and reasonable to do so.

5.1. Sequences

A sequence of two or more code points may be specified in a LGR,, for example, when defining the source for n:m variant mappings. Another use of sequences would be in cases when the exact sequence of code points is required to occur in order for the constituent elements to be eligible, such as when a specific code point is only eligible when preceded or followed by another code point. The following would define the eligibility of the MIDDLE DOT (U+00B7) only when both preceded and followed by the LATIN SMALL LETTER L (U+006C):

```
<char cp="006C 00B7 006C" comment="Catalan middle dot"/>
```

As an alternative to using sequences to define a required context, a "char" or "range" element may specify conditional context in a "when" attribute as described below in Section 5.2.5. The latter method is more flexible in that such conditional context is not limited to specific code point. and allows prohibited, as well as required context to be specified.

5.2. Variants

While most LGRs typically only determine code point eligibility, others additionally specify a mapping of code points to other code points, known as "variants". What constitutes a variant is a matter of policy, and varies for each implementation. The following examples are intended to demonstrate the syntax; they are not necessarily typical.

5.2.1. Basic variants

Variants are specified using one of more "var" elements as children of a "char" element.

For example, to map LATIN SMALL LETTER V (U+0076) as a variant of LATIN SMALL LETTER U (U+0075):

```
<char cp="0075">
    <var cp="0076"/>
</char>
```

A sequence of multiple code points can be specified as a variant of a single code point. For example, the sequence of LATIN SMALL LETTER O (U+006F) then LATIN SMALL LETTER E (U+0065) might hypothetically be specified as a variant for an LATIN SMALL LETTER O WITH DIAERESIS (U+00F6) as follows:

```
<char cp="00F6">
    <var cp="006F 0065"/>
</char>
```

The "var" element specifies Variants in only one direction, even though the variant relation is usually considered symmetric, that is, if A is a variant of B then B is typically also a variant of A. The format requires that the inverse of the variant is given explicitly to fully specify symmetric variant relations in the IDN table. This has the beneficial side effect of making the symmetry explicit:

```
<char cp="006F 0065">
    <var cp="00F6"/>
</char>
```

Both the source and target of a variant mapping may be sequences. As it is not possible to specify variants for ranges, ranges cannot be used for characters for which variant relations need to be defined.

5.2.2. Null variants

To specify a null variant, which is a variant string that maps to no code point, use an empty cp attribute. For example, to mark a string with a ZERO WIDTH NON-JOINER (U+200C) to the same string without the ZERO WIDTH NON-JOINER:

```
<char cp="200C">
    <var cp=""/>
</char>
```

The symmetric form of a null variant cannot be expressed in the IDN table format.

5.2.3. Dispositions

Variants may be given dispositions. These describe the policy state for a variant label that was generated using a particular variant. The dispositions are the same as described below in Section 6.4.

A disposition may be of any value, but several conventional dispositions are predefined below in Section 6.4 to encourage common conventions in their application. If these values can represent registry policy, they SHOULD be used.

```
<char cp="767C">
    <var cp="53D1" disposition="allocate"/>
    <var cp="5F42" disposition="block"/>
    <var cp="9AEA" disposition="block"/>
    <var cp="9AEE" disposition="block"/>
</char>
```

Usually, if a variant label contains any instance of one of the blocked variants the label would be blocked, but if it contained only instances of allocated variants it could be allocated. See the discussion about implied actions in Section 6.4.3.

5.2.4. The ref attribute

Reference information may optionally be specified by a "ref" attribute, consisting of a space delimited sequence of reference identifiers.

```
<char cp="522A" ref="0">
    <var cp="5220" ref="2 3"/>
    <var cp="5220" ref="2 3"/>
</char>
```

This facility is typically used to give source information for characters or variant relations. This information is ignored when machine-processing an LGR. Specifying a "ref" attribute on a range element is equivalent to specifying the same ref attribute on every single code point of the range. The reference identifiers MUST match those declared in the "references" element (see Section 4.3.8).

In addition to "char", "range" and "var" elements in the data section, a ref attribute may be present for literals ("char" inside a rule) as well as rules and class definitions, but not for named references to them.

5.2.5. Conditional variants

Fundamentally, variants are mappings between two sequences of code points. However, in some instances for a variant relationship to exist, some context external to the code point sequence must be considered. For example, in some cases the positional context determines whether two code point sequences are variants of each other. An example are the Arabic characters, which can have different forms based on position. This positional context cannot be solely derived from the code point, as the code point is the same for the various forms.

To specify a conditional variant relationship the "when" attribute is used. The variant relationship exists when the condition in the "when" attribute is satisfied. A "not-when" attribute may be used for conditions that must not be satisfied. The value of each "when" or "not-when" attributes is a a parameterized context rule as described below inSection 6.3.2.

Assuming the "rules" element contains suitably defined rules for "arabic-isolated" and "arabic-final", the following example shows how to mark ARABIC LETTER ALEF WITH WAVY HAMZA BELOW (U+0673) as a variant of ARABIC LETTER ALEF WITH HAMZA BELOW (U+0625), but only when it appears in isolated or final forms:

```
<char cp="0625">
    <var cp="0673" when="arabic-isolated"/>
    <var cp="0673" when="arabic-final"/>
</char>
```

Only a single "when" or "not-when" attribute can be applied to any

"var" element, however, multiple "var" elements using the same mapping, but different "when" or "not-when" attributes may be specified.

While currently Arabic is the only script known for which such conditional variants are defined. there are other scripts, such as Mongolian, which share the concept of positional forms. By requiring explicit definitions for these rules, this mechanism can easily handle any additional types of conditional variants that are required.

As described in Section 5.1 a "when" or "not-when" attribute may also be specified to any "char" element in the data section to define required or prohibited contextual conditions under which a code point is valid.

5.2.6. The comment attribute

Any "char", "range" or "variant" element may contain a comment in a "comment" attribute. The contents of a comment attribute are free-form plain text. Comments are ignored in machine processing of the table. Comment attributes may also be placed on certain elements in the "rules" section of the document, such as actions and literals ("char"), as well as definitions of classes and rules, but not named references to them. Finally, in the metadata the "version" and "reference" elements may have comment attributes to match the syntax in [RFC3743]

5.3. Code point tagging

Typically, LGRs are used to explicitly designate allowable code points, with any label with a code point not explicitly listed in the LGR being considered an ineligible label according to the ruleset.

For more complex registry rules, there may be a need to discern code points of certain types. This can be accomplished by applying a "tag" attribute, and then filtering on results based on the tag using whole label evaluation. Tag attributes may be of any value, and multiple values are separated by space.

A simple example would be to label preferred code points (as in[RFC3743]) by adding "preferred" to the tag, and then using a rule such as shown in Section 6.3.1 to filter out labels that consist entirely of such preferred code points.

6. Whole Label and Context Evaluation

6.1. Basic concepts

The code points in a label sometimes need to satisfy context-based rules, for example for the label to be considered valid, or to satisfy the context for a variant mapping (see the description of the "when" attribute in Section 6.3.2).

A Whole Label Evaluation rule (WLE) is applied to the whole label. It is used to validate both original labels and variant labels derived from them. A conditional context rules is a specialized form of WLE specific to the context around a single code point or code point sequence. For example, if a rule is referenced in the "when" attribute of a variant mapping it is used to describe the conditional context under which the particular variant mapping is defined to exist.

Each rule is defined in a "rule" element. A rule may contain the following as child elements:

- literal code points or code point sequences
- character classes, which defines sets of code points to be used for context comparisons; and
- context operators, which define when character classes and literals may appear

6.2. Character Classes

Character classes are sets of characters, that often share a particular property. They can be specified in several ways:

1. by defining the property via matching a tag in the code point data. All characters with the same tag attribute are part of the same class.
2. by referencing one of the Unicode character properties defined in the Unicode Character Database[UAX42];
3. by explicitly listing all the code points in the class; or
4. by defining the class as a combination of any number of these definitions or other classes.

A character class has an optional "name" attribute, consisting of a single identifier not containing spaces. If it is omitted, the class

is anonymous and exists only inside the rule or combined class where it is defined. A named character class is defined independently and can be referenced by name by both rules and other character classes.

```
<class name="example" comment="an example class definition">
    <char cp="0061" />
    <char cp="4E00" />
</class>
...
<rule>
    <class name="example" />
</rule>
```

An empty "class" element with a name attribute is a reference to an existing named class. Such an element MUST not have either "comment" or "ref" attributes as those may only be placed on a class definition.

6.2.1. Tag-based classes

The char element may contain a tag attribute that consists of one or more space separated identifiers, for example:

```
<char cp="0061" tag="letter lower"/>
<char cp="4E00" tag="letter"/>
```

This defines two tags for use with code point U+0061, the tag "letter" and the tag "lower". Implicitly, this defines two named character classes, the class "letter" and the class "lower", the first with 0061 and 4E00 as elements and the latter with 0061, but not 4E00 as an element. The document MUST not contain an explicitly named class definition of the same name as an implicitly named tag-derived class.

6.2.2. Unicode property based classes

A class is defined in terms of Unicode properties by giving the Unicode property alias and the property value or property value alias, separated by a colon.

```
<class name="virama" property="ccc:9" />
```

The example above selects all characters for which the Unicode canonical combining class (ccc) value is 9. This value of the ccc is assigned in the to all characters that are viramas. The string "ccc" is the short-alias for the canonical combining class, as defined in the Unicode Character Database [UAX42].

Unicode properties may, in principle, change between versions of the Unicode Standard. However, the values assigned for a given version are fixed. If Unicode Properties are used, a minimum Unicode version MUST be declared in the header. (Note, some Unicode properties are by definition stable across versions and do not change once assigned.)

6.2.3. Explicitly declared classes

A class of code points may also be declared by listing the code points that are a member of the class. This is useful when tagging cannot be used because code points are not listed individually as part of the eligible set of code points for the given LGR, for example because they only occur in code point sequences.

To define a class in terms of an explicit list of code points:

```
    <class name="abc">
        <char cp="0061"/>
        <char cp="0062"/>
        <char cp="0063"/>
   </class>
```

This defines a class named "abc" containing the code points for characters "a", "b" and "c". The ordering of the code points is not material, but it is RECOMMENDED to list them in ascending order.

Range operators may also be used to represent any series of consecutive code points. The same declaration can be made as follows:

```
    <class name="abc">
        <range first-cp="0061" last-cp="0063"/>
    </class>
```

Range and code point declarations can be freely intermixed. A shorthand notation exists where code points are directly represented by space separated hexadecimal values, and ranges are represented by a start and end value separated by a hyphen.

```
    <class name="abc">0061 0062-0063</class>
```

would be a more streamlined expression of the same class using the shorthand notation.

6.2.4. Combined classes

Classes may be combined using logical operators for inversion, union, intersection, difference and symmetric difference (exclusive-or).

Logical Operation	Example
Inversion	<not><class name="xxx"></not>
Union	<union> <class name="class-1"/> <class name="class-2"/> </union>
Intersection	<intersection> <class name="class-1"/> <class name="class-2"/> </intersection>
Difference	<difference> <class name="class-1"/> <class name="class-2"/> </difference>
Symmetric Difference	<symmetric-difference> <class name="class-1"/> <class name="class-2"/> </symmetric-difference>

Combinations can be anonymous or named.

```
<union name="xxxyyy">
    <class name="xxx"/>
    <class name="yyy"/>
</union>
```

This creates a named class that represents the union of classes "xxx" and "yyy", and which can be referenced in other classes or rules as <class name="xxxyyy"/>.

Note that the reference to a named class is always via a "class" element, independent of how the character class was defined.

An "intersection", "symmetric-difference" or "difference" element MUST contain precisely two, and a "not" element MUST contain

precisely one "class" or one of the operator elements, while a "union" element MUST contain two or more elements.

6.3. Whole Label and Context Rules

Each rule is comprised of a series of matching operators that must be satisfied in order to determine whether a label meets a given condition. Rules may reference other rules or character classes defined elsewhere in the table.

6.3.1. The rule element

A matching rule is defined by a "rule" element, which contains combinations character classes with literal code point sequences and context operators contained in child elements. In evaluating a rule, each child element is matched in order.

Rules may optionally be named using a "name" attribute containing a single identifier string with no spaces. If the name attribute is omitted, the rule is anonymous and may not incorporated by reference into another rule or referenced by an action or "when" attribute.

A simple rule to match a label where all characters are members of the class "preferred":

```
<rule name="preferred" match="whole-label">
    <class name="preferred" count="1+"/>
</rule>
```

Rules are paired with explicit and implied actions, triggering these actions when a rule matches a label. For example, a simple explicit action for the rule shown above would be:

```
<action disposition="preferred" match="preferred" />
```

which has the effect of setting the policy disposition for a label made up entirely of "preferred" code points, to "preferred". Explicit actions are further discussed in Section 6.4 and use of rules in conditional context for implied actions is discussed in Section 5.2.5 and Section 6.4.3.

6.3.1.1. The count attribute

The number of times a specific character class or rule may appear in an expression defined by a rule is given by the "count" attribute. The attribute consists of a number, optionally followed by a "+" sign. The number MUST be an integer of value 0 or higher, and gives the number of times the class or rule may appear in matching. If the

number is followed by a plus sign ("+"), it means that any number of additional occurrences are allowed beyond the number stated. Therefore, "1" would mean exactly one occurrence, whereas "1+" would indicate one or more occurrences.

If no count attribute is specified, the number of occurrences is "1".

6.3.1.2. The choice element

For cases where several alternates could be chosen, the "choice" element can encode a list of choices:

```
<rule name="ldh">
   <choice count="1+">
       <class name="letters"/>
       <class name="digits"/>
       <char cp="002D"/>
   </choice>
</rule>
```

Each child element of a "choice" represents one alternative. The first matching alternative determines the match for the choice element. To express a choice where one alternative consists of a sequence of elements, they can be wrapped in an anonymous rule.

6.3.1.3. Literal code point sequences

A literal code point sequence matches a single code point or a sequence. It is defined by a "char" element, with the code point or sequence to be matched given by the "cp" attribute. When used as a literal, a "char" element may contain a "count" in addition to the "cp" attribute, comments or references, but no conditional contexts or child elements.

6.3.1.4. The any element

The "any" element matches any single code point. It may have a "count" attribute. For an example see Section 6.3.1.8

The "any" element" may have neither a "comment" nor a "ref" attribute.

6.3.1.5. The start and end elements

To match the beginning or end of a label, use the "start" or "end" element.

```
<rule name="empty-label">
    <start/>
    <end/>
</rule>
```

Start and end elements do not have a "count" or any other attribute. When their use is not required, it is RECOMMENDED to us the "match" attribute instead. One case where start or end elements are required is when only some, but not all of the alternatives in a "choice" need to match beginning or end of a label.

6.3.1.6. The match attribute

Whole Label Evaluation Rules in principle always apply to the entire label, but in practice, for example to express a requirement to not start a label with a digit, some rules do not need to cover the whole label. Use attribute "match" with value "whole-label" to identify a rule applicable to the entire label. For other rules use "from-start", "anywhere" and "to-end" to define rules that need to match in specific positions of the label. Certain parameterized context rules (see Section 6.3.2) have a match attribute value of "context". The defaults is "anywhere".

A "match" attribute present in the definition of a rule is ignored if a rule is referenced by name inside another rule. An anonymous rule may not have a "match" attribute.

6.3.1.7. The name attribute

Rules and classes may be named using a "name" attribute and can be nested either directly or, if named, by reference.

Here's an example of a rule requiring that all labels be letters (optionally followed by combining marks) and possibly digits. The example shows rules and classes referenced by name.

```
<class name="letter" property="gc:L"/>
<class name="combining-mark" property="gc:M"/>
<class name="digit" property="gc:Nd">
<rule name="letter-grapheme">
   <class name="letter" count="1+"/>
   <class name="combining-mark" count="0+"/>
</rule>
<rule name="leading-letter" match="whole-label">
   <rule name="letter-grapheme" count="1"/>
   <choice count="0+">
       <rule name="letter-grapheme" count="0+"/>
       <class name="digit" count="0+"/>
   </choice>
</rule>
```

6.3.1.8. Example rule from IDNA2008

This sections shows an example of the whole label evaluation rule from[RFC5892]forbidding the mixture of the Arabic-Indic and extended Arabic-Indic digits in the same label.

```
<data>
    <range first-cp="0660" last-cp="0669" not-when="mixed-digits"
         tag="arabic-indic-digits" />
    <range first-cp="06F0" last-cp="06F9" not-when="mixed-digits"
         tag="extended-arabic-indic-digits" />
</data>
<rules>
<rule name="mixed-digits" match="anywhere">
    <choice>
        <rule>
            <class name="arabic-indic-digits"/>
            <any count="0+"/>
            <class name="extended-arabic-indic-digits"/>
        </rule>
        <rule>
            <class name="extended-arabic-indic-digits"/>
            <any count="0+"/>
            <class name="arabic-indic-digits"/>
        </rule>
    </choice>
</rule>
```

The preceding example also demonstrates several instances of the use of anonymous rules for grouping.

6.3.2. Parameterized Context or When Rule

A special type of rule provides a context for evaluating the validity of a code point or variant mapping. This rule is invoked by the "when" attribute described in Section 5.2.5. For a context rule, the match attribute is normally "context". Such "when rules" contain a special place holder, represented by a "match" element (not to be confused with the "match" attribute). When evaluated, the "match" element is replaced by a literal corresponding to the "cp" attribute of the element for which the rule in its "when" attribute is being evaluated.

For example, the Greek lower numeral sign is invalid if not immediately preceding a character in the Greek script. This is most naturally addressed with a when rule using look-ahead:

```
<char cp="0375" when="preceding-greek"/>
...
<class name="greek-script" property="sc:Grek"/>
<rule name="preceding-greek" match="context">
    <match/>
    <look-ahead>
        <class name="greek-script"/>
    </look-ahead>
</rule>
```

In evaluating this rule, the "match" element is treated as if it was replaced by a literal

```
<char cp="0375"/>
```

The action implied by a context rule is always a disposition of "invalid" if the when rule is not matched. Unlike other rules, these rules may not be associated with arbitrary actions via "action" elements.

6.3.2.1. The look-behind and look-ahead elements

Context rules use the "look-behind" and "look-ahead" elements to define context before and after the code point sequence matched by the "match" element. If the "match" element is omitted, neither the "look-behind" nor the "look-ahead" element may be present.

Here is an example of a rule that defines an "initial" context for an Arabic code point:

```
<class name="transparent" property="jt:T"/>
<class name="right-joining" property="jt:R"/>
<class name="left-joining" property="jt:L"/>
<class name="dual-joining" property="jt:D"/>
<class name="non-joining" property="jt:U"/>
<rule name="Arabic-initial" match="context">
  <look-behind>
    <choice>
      <start/>
      <rule>
        <class name="transparent" count="0+"/>
        <class name="non-joining"/>
      </rule>
    </choice>
  </look-behind>
  <match />
  <look-ahead>
    <class name="transparent" count="0+" />
    <choice>
      <class name="right-joining" />
      <class name="dual-joining" />
    </choice>
  </look ahead>
</rule>
```

A when rule contains any combination of "look-ahead" , "match" and "look-behind" elements in that order. Each of these elements occurs at most once, and none have a "count" attribute. If a context rule contains a look-ahead or look-behind element, it MUST contain a "match" element. If a "match" element is present the rule MUST have a "match" attribute with a value of "context".

6.3.2.2. Omitting the match element

If the "match" element is omitted, the evaluation of the context rule is not tied to the position of the code point or sequence associated with the "when" attribute.

Katakana middle dot is invalid in any label not containing at least one Japanese character anywhere in the label. Because this requirement is independent of the position of the middle dot, the rule does not require a "match" element and the "match" attribute is "anywhere".

```
    <char cp="30FB" when="japanese-in-label"/>
    <rule name="japanese-in-label" match="anywhere">
        <union>
            <class property="sc:Hani"/>
            <class property="sc:Kata"/>
            <class property="sc:Hira"/>
        </union>
    </rule>
```

The Katakana middle dot is used only with Han, Katakana or Hiragana. The "when" rule requires that at least one code point in the label is in one of these scripts. (Note that the Katakana middle dot itself is of script Common).

6.4. Action elements

The purpose of a rule is to trigger a specific action. Often, the action simply results in blocking a label that does not match a rule. An example of an action invalidating a label:

```
<action disposition="invalid" not-match="leading-letter"/>
```

An action may contain precisely one "match" or "not-match" attribute, but not both. Because rules may be compound rules that contain other rules, only a single rule may be named as the value of the "match" or "not-match" attribute.

An action may contain either one of a set of optional attributes matching the variant disposition from the "disposition" attributed of any "var" element used in generating the variant label being evaluated. Assuming all variants have been given suitable "disposition" attributes of "blocked" or "allocate" and that a rule is defined matching labels consisting entirely of code points tagged as "preferred" the following actions evaluate the disposition for the variant label:

```
    <action disposition="blocked" any-variant="blocked" />
    <action disposition="activate" all-variants="allocate"
        match="preferred" />
```

The first action matches any variant label for which at least one of the code point variants carries the disposition "blocked". The second matches any variant label for which all of the code point variants carry the disposition "allocate". Neither action matches a label that is not a variant label. If necessary repeat an action so it applies to an ordinary label:

```
    <action disposition="activate"  match="preferred" />
```

6.4.1. Recommended Disposition Values

The precise nature of the policy action taken in response to a disposition and the name of the corresponding "disposition" attributes are only partially defined here. It is strongly RECOMMENDED to use the following actions only with their conventional sense.

invalid The resulting string is not a valid label. This disposition may be assigned implicitly, see Section 6.4.3.

block The resulting string is a valid label, but should be blocked from registration. This would typically apply for a derived variant that has no practical use, such as blocking confusingly similar by undesirable variants.

allocate The resulting string should be reserved for use by the same operator of the origin string, but not automatically allocated for use.

activate The resulting string should be activated for use. (This is the typical default action if no tagging is used, and is known as a "preferred" variant in [RFC3743])

6.4.2. Precedence

Actions are applied in the order of their appearance in the file. This defines their relative precedence. The first action for which the rule is matched or not-matched as required for a particular label defines the disposition for that label. The conventional order of precedence for the actions defined here is "invalid", "block", "allocate", "activate". In order to define a different order of precedence or when additional actions are defined, list the actions in the appropriate order.

6.4.3. Implied actions

The context, or "when" rules carry an implied action with a disposition of "invalid". These rules are evaluated at the time a label's code points and variants are checked for validity (see Section 8) . In other words, before any whole-label evaluation rules and with higher precedence. The context rules for variant mappings are evaluated when variants are generated and / or when variant tables are made symmetric and transitive. They have an implied action with a disposition of "invalid" which means a putative variant mapping doesn't exist in the given context.

Note that such non-existing variant mapping is different from a

blocked variant, which is variant code point mapping that exists, but results in a label that may not be allocated.

Variant mappings may be given a disposition attribute . An implied action relates these to the disposition for the entire variant label. For example, a variant label in which any variant code point is a blocked code point variant is blocked. The default order of precedence for evaluating dispositions is as given above. The default precedence applies if no actions are defined that match specific variant dispositions.

7. Example table

The following presents a sample XML LGR showing a near complete collection of most of the elements and attributes defined in this specification in somewhat typical context.

```
<?xml version="1.0" encoding="utf-8"?>
<lgr xmlns="http://www.iana.org/lgr/0.1">

  <meta>
    <version>1</version>
    <date>2010-01-01</date>
    <language>sv</language>
    <domain>example</domain>
    <description type="text/html">
        <![CDATA[
        This language table was developed with the
        <a href="http://swedish.example/">Swedish
        examples institute</a>.
        ]]>
    </description>
    <references>
      <reference id="0" >The Unicode Standard 6.3</reference>
      <reference id="1" >RFC 5892</reference>
      <reference id="2" >Big-5: Computer Chinese Glyph and Character
         Code Mapping Table, Technical Report C-26, 1984</reference>
    </references>
 </meta>
  <data>
    <char cp="002D" ref="1" comment="HYPHEN" />
    <range first-cp="0030" last-cp="0039" ref="1" tag="digit" />
    <range  first-cp="0061" last-cp="007A" ref ="1" tag="letter" />
    <range first-cp="0370" last-cp="0380"  />
    <char cp="00B7" when="catalan-middle-dot" />
    <char cp="200D" when="joiner" />
    <char cp="4E16" tag="preferred" ref="0">
      <var cp="4E17" disposition="blocked" ref="2" />
      <var cp="534B" disposition="allocate" ref="2" />
    </char>
    <char cp="4E17" ref="0">
      <var cp="4E16" disposition="allocate" ref="2" />
      <var cp="534B" disposition="allocate" ref="2" />
    </char>
    <char cp="534B" ref="0">
      <var cp="4E16" disposition="allocate" ref="2" />
      <var cp="4E17" disposition="blocked" ref="2" />
    </char>
  </data>
```

```
    <rules>
      <class name="virama" property="ccc:9" />
      <rule name="catalan-middle-dot" match="context" ref="0">
          <look-behind>
              <char cp="006C" />
          </look-behind>
          <match />
          <look-ahead>
              <char cp="006C" />
          </look-ahead>
      </rule>
      <rule name="joiner" match="context" ref="1" >
          <look-behind>
              <class name="virama" />
          </look-behind>
      </rule>
      <rule name="example" >
          <difference>
              <not>
                  <class comment="use shorthand class notation">
                      006E 0070-0078
                  </class>
              </not>
              <class comment="use standard notation">
                  <range first-cp="0000" last-cp="001F" />
                  <char cp="007F" />
              </class>
          </difference>
      </rule>
      <rule name="preferred"
            comment="non-empty lablel of preferred code points">
          <class name="preferred" count="1+" />
      </rule>
      <action disposition="example" match="example" />
      <action disposition="blocked" any-variant="blocked" />
      <action disposition="activate" all-variants="allocate"
            match="preferred" />
      <action disposition="activate"  match="preferred" />
    </rules>
  </lgr>
```

8. Processing a label against an LGR

8.1. Determining eligibility for a label

In order to use a table to test a specific domain label for membership in the LGR, a consumer of the LGR must iterate through each code point within a given U-label, and test that each code point is a member of the LGR. If any code point is not a member of the LGR, it shall be deemed as not eligible in accordance with the table.

A code point is deemed a member of the table when it is listed with the "char" element, and all necessary condition listed in "when" or "not-when" attributes are correctly satisfied.

8.2. Determining variants for a label

For a given eligible label, the set of variants is deemed to be each possible permutation of "var" elements, whereby all "when" and "not-when" attributes are correctly satisfied for each var element in the given permutation and all applicable whole label evaluation rules are satisfied as follows:

- Create each possible permutation of a label, by substituting each code point or code point sequence in turn by any defined variant mapping

- Apply variant mappings with "when" or "not-when" attributes only if the conditions are satisfied

- Record each of the "disposition" values on the variant mappings used in creating a given variant label

- Evaluate each variant against any actions for which the disposition is "invalid", remove any that satisfy the conditions.

8.3. Determining a disposition for a label or variant label

For a given label, the disposition for the is determined by evaluating in order of their appearance all actions for which the label or variant label satisfies the conditions.

- For any label, the "disposition" value for the first action applies, for which the label matches or doesn't match the whole label evaluation rule, given in the "match" or "not-match" attribute for that action,

- For any variant label, the "disposition" value for the first action applies, for which the label matches or doesn't match the

whole label evaluation rule, given in the "match" or "not-match" attribute, and for which any or all of the recorded variant dispositions match the conditions for that action.

- For any remaining variant label, assign the variant label the disposition matching the most restrictive disposition recorded. for any of its variants The order from most restrictive to least is "invalid", "blocked", "allocated", "active".

- Variants dispositions outside the predefined default set, and for which no action is defined are ignored.

9. Conversion between other formats

Both [RFC3743] and [RFC4290] provide different grammars for IDN tables. These formats are unable to fully cater for the increased requirements of contemporary IDN variant policies.

This specification is a superset of functionality provided by these IDN table formats, thus any table expressed in those formats can be expressed in this format. Automated conversion can be conducted between tables conformant with the grammar specified in each document.

10. IANA Considerations

This document does not specify any IANA actions.

11. Security Considerations

There are no security considerations for this memo.

12. References

[LGR-PROCEDURE]
Internet Corporation for Assigned Names and Numbers, "Procedure to Develop and Maintain the Label Generation Rules for the Root Zone in Respect of IDNA Labels".

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC3743] Konishi, K., Huang, K., Qian, H., and Y. Ko, "Joint Engineering Team (JET) Guidelines for Internationalized Domain Names (IDN) Registration and Administration for Chinese, Japanese, and Korean", RFC 3743, April 2004.

[RFC4290] Klensin, J., "Suggested Practices for Registration of Internationalized Domain Names (IDN)", RFC 4290, December 2005.

[RFC5564] El-Sherbiny, A., Farah, M., Oueichek, I., and A. Al-Zoman, "Linguistic Guidelines for the Use of the Arabic Language in Internet Domains", RFC 5564, February 2010.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[UAX42] Unicode Consortium, "Unicode Character Database in XML".

[XML] "Extensible Markup Language (XML) 1.0".

Appendix A. RelaxNG Schema

[TODO: this needs to be updated to reflect additions to the syntax.]

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar ns="http://www.iana.org/lgr/0.1"
  xmlns="http://relaxng.org/ns/structure/1.0">
  <!-- SIMPLE TYPES -->
  <define name="language-tag">
    <text/>
  </define>
  <!-- RFC 5646 language tag (e.g. "de", "Latn", etc.) -->
  <define name="domain-name">
    <text/>
  </define>
  <!-- Domain name -->
  <define name="code-point">
    <text/>
  </define>
  <!-- A single codepoint, expressed as a hexidecimal number -->
  <define name="variant-condition">
    <text/>
  </define>
  <!-- A condition for applying the variant (TBD) -->
  <define name="tag">
    <text/>
  </define>
  <!-- Freeform text tag -->
  <!-- STRUCTURES -->
  <!-- Representation of a single codepoint -->
  <define name="point-single">
    <element name="char">
      <attribute name="cp">
        <ref name="code-point"/>
      </attribute>
      <attribute name="tag">
        <ref name="tag"/>
      </attribute>
      <optional>
        <attribute name="ref"/>
      </optional>
      <zeroOrMore>
        <ref name="point-variant"/>
      </zeroOrMore>
    </element>
  </define>
  <!-- Representation of a code point variant -->
```

```
    <define name="point-variant">
      <element name="var">
        <attribute name="cp">
          <ref name="code-point"/>
        </attribute>
        <optional>
          <attribute name="type"/>
        </optional>
        <optional>
          <attribute name="when">
            <ref name="variant-condition"/>
          </attribute>
        </optional>
        <optional>
          <attribute name="comment"/>
        </optional>
        <optional>
          <attribute name="disposition"/>
        </optional>
        <optional>
          <attribute name="ref"/>
        </optional>
      </element>
    </define>
    <!-- Representation of a range of codepoints -->
    <define name="point-multiple">
      <element name="range">
        <attribute name="first-cp">
          <ref name="code-point"/>
        </attribute>
        <attribute name="last-cp">
          <ref name="code-point"/>
        </attribute>
        <text/>
      </element>
    </define>
    <define name="logical-operators">
      <choice>
        <element name="not">
          <ref name="class-points"/>
        </element>
        <element name="union">
          <oneOrMore>
            <ref name="class-points"/>
          </oneOrMore>
        </element>
        <element name="intersection">
          <oneOrMore>
```

```
            <ref name="class-points"/>
          </oneOrMore>
        </element>
        <element name="difference">
          <oneOrMore>
            <ref name="class-points"/>
          </oneOrMore>
        </element>
        <element name="symmetric-difference">
          <oneOrMore>
            <ref name="class-points"/>
          </oneOrMore>
        </element>
      </choice>
    </define>
    <!--
      A collection of codepoints and ranges of codepoints that comprise
      a label generation ruleset
    -->
    <define name="points">
      <oneOrMore>
        <choice>
          <ref name="point-single"/>
          <ref name="point-multiple"/>
        </choice>
      </oneOrMore>
    </define>
    <define name="class-points">
      <choice>
        <ref name="point-single"/>
        <ref name="point-multiple"/>
        <ref name="logical-operators"/>
      </choice>
    </define>
    <define name="any">
      <element name="any">
        <optional>
          <attribute name="count"/>
        </optional>
      </element>
    </define>
    <define name="class">
      <element name="class">
        <optional>
          <attribute name="count"/>
        </optional>
        <optional>
          <attribute name="name"/>
```

```
        </optional>
        <optional>
          <attribute name="comment"/>
        </optional>
        <optional>
          <attribute name="ref"/>
        </optional>
        <optional>
          <attribute name="property"/>
        </optional>
        <choice>
          <ref name="class-points"/>
          <text/>
        </choice>
      </element>
    </define>
    <define name="choice">
      <element name="choice">
        <optional>
          <attribute name="count"/>
        </optional>
        <oneOrMore>
          <ref name="class-matchers"/>
        </oneOrMore>
      </element>
    </define>
    <define name="class-matchers">
      <oneOrMore>
        <choice>
          <ref name="class"/>
          <ref name="any"/>
          <ref name="choice"/>
        </choice>
      </oneOrMore>
    </define>
    <define name="rules-declaration">
      <element name="rule">
        <attribute name="name"/>
        <oneOrMore>
          <ref name="class-matchers"/>
        </oneOrMore>
      </element>
    </define>
    <define name="action-declaration">
      <element name="action">
        <attribute name="action"/>
        <choice>
          <attribute name="match"/>
```

```
          <attribute name="not-match"/>
        </choice>
      </element>
    </define>
    <!-- DOCUMENT STRUCTURE -->
    <!--
      Main document structure, comprised of a meta section followed by
      a data section.
    -->
    <start>
      <ref name="lgr"/>
    </start>
    <define name="lgr">
      <element name="lgr">
        <attribute name="id"/>
        <optional>
          <ref name="meta-section"/>
        </optional>
        <ref name="data-section"/>
        <optional>
          <ref name="rules-section"/>
        </optional>
      </element>
    </define>
    <!--
      Meta section - information recorded with an label
      generation ruleset that does not affect machine processing.
    -->
    <define name="meta-section">
      <element name="meta">
        <zeroOrMore>
          <choice>
            <optional>
              <element name="version">
                <text/>
              </element>
            </optional>
            <optional>
              <element name="date">
                <text/>
              </element>
            </optional>
            <zeroOrMore>
              <element name="language">
                <ref name="language-tag"/>
              </element>
            </zeroOrMore>
            <zeroOrMore>
```

```
              <element name="domain">
                <ref name="domain-name"/>
              </element>
            </zeroOrMore>
            <optional>
              <element name="validity-start">
                <text/>
              </element>
            </optional>
            <optional>
              <element name="validity-end">
                <text/>
              </element>
            </optional>
            <optional>
              <element name="unicode-version">
                <text/>
              </element>
            </optional>
            <zeroOrMore>
              <element name="description">
                <attribute name="type"/>
                <text/>
              </element>
            </zeroOrMore>
            <optional>
              <element name="references">
                <zeroOrMore>
                  <element name="reference">
                    <attribute name="id"/>
                    <text/>
                  </element>
                </zeroOrMore>
              </element>
            </optional>
          </choice>
        </zeroOrMore>
      </element>
    </define>
    <!-- Data section - the actual code point data of the table. -->
    <define name="data-section">
      <element name="data">
        <ref name="points"/>
      </element>
    </define>
    <!-- Rules section -->
    <define name="rules-section">
      <element name="rules">
```

```
        <zeroOrMore>
          <choice>
            <ref name="rule-declaration"/>
            <ref name="action-declaration"/>
          </choice>
        </zeroOrMore>
      </element>
    </define>
  </grammar>
```

Appendix B. Acknowledgements

This format builds upon the work on documenting IDN tables by many different registry operators. Notably, a comprehensive language table for Chinese, Japanese and Korean was developed by the "Joint Engineering Team" [RFC3743] that is the basis of many registry policies; and a set of guidelines for Arabic script registrations [RFC5564] was published by the Arabic-language community.

Contributions that have shaped this document have been provided by Francisco Arias, Mark Davis, Nicholas Ostler, Thomas Roessler, Steve Sheng, Michel Suignard, John Yunker and Andrew Sullivan.

Appendix C. Editorial Notes

This appendix to be removed prior to final publication.

C.1. Known Issues and Future Work

- A method of specifying the origin URI for a table, and an expiration or refresh policy, as meta-data may be a useful way to declare how the table will be updated.

C.2. Change History

-00 Initial draft.

-01 Add an XML Namespace, and fix other XML nits. Add support for sequences of code points. Improve on consistently using Unicode nomenclature.

-02 Add support for validity periods.

-03 Incorporate requirements from the Label Generation Ruleset Procedure for the DNS Root Zone. These requirements include a detailed grammar for specifying whole-label variants, and the ability to explicitly declare of the actions associated with a specific variant. The document also consistently applies the term "Label Generation Ruleset", rather than "IDN table", to reflect the policy term now being used to describe these.

-04 Support reference information per [RFC3743]. Update description in response to feedback. Extend the context rules to "char" elements and allow for inverse matching ("not-when"). Extend the description of label processing and implied actions, and allow for actions that reference disposition attributes on any or all variant mappings used in the generation of a variant label.

Authors' Addresses

Kim Davies
Internet Corporation for Assigned Names and Numbers
12025 Waterfront Drive
Los Angeles, CA 90094
US

Phone: +1 310 301 5800
Email: kim.davies@icann.org
URI: http://www.iana.org/

Asmus Freytag
ASMUS Inc.

Email: asmus@unicode.org