Reference Label Generation Rules (LGR) for the Second Level — Overview and Summary

REVISION – January 24, 2023

1 Overview

This document describes a set of Reference Label Generation Rules (LGRs) for the Second Level. These reference LGRs were developed according to the “Guidelines for Developing Reference LGRs for the Second Level” [Guidelines]. The set divides into *language-based* and *script-based* reference LGRs. There are some differences in the origin, methodology and design goals between these sets.

Reference LGRs are intended as a starting point for the actual LGRs adopted by a registry for a given zone. In many cases, they can be adopted as they are, with only minor changes to the published files.

The development of a large part of the language-based LGRs builds on the results of a previous project [IIS] but provides additional review and development, documentation, and translation to XML [RFC7940]. Where indicated, this may result in extending the repertoire compared to [IIS]. All other LGRs, including script LGRs, are derived from the Root Zone LGR for the corresponding script with suitable additions for the second level and incorporating additional input from the same panels that developed the Root Zone LGRs. Language LGRs derived from Root Zone LGRs have a repertoire that is based on the Root Zone LGR, but restricted to specific languages. Each reference LGR contains a description section that documents any issues specific to that LGR. Each also contains a set of instructions for adopting the LGR for a specific zone and for preparing it for deposit in the IANA Repository [IANA].

The reference LGRs are specific to a given language or script (and in some cases the combination of language and script) but not necessarily specific to a geographically compact user community. Each file has been reviewed by the community and some have seen additional review by one or more linguistic experts, as well as reviewed by a separate expert for DNS stability and security issues.

Some of these LGRs are more suitable for standalone use, with the TLD supporting a single language or script (see Section 1.2.1 “LGRs Omitting Cross-Script Variant Listings” below). Other LGRs have features that make it easier to support them in combination with one or more reference LGRs other than those intended for standalone use. A *Common LGR* has been created by merging the data from the script-based LGRs. This file is intended for collision checking, particularly in the case where multiple LGRs are used in the same zone.

This *overview document* provides some general background related to the design and design process common to these reference LGRs, as well as general considerations relevant to anyone wishing to adopt or adapt these LGRs for use in a particular zone. Not all reference LGRs may have been developed or updated on the same schedule; this document applies to all that have been published or are actively under development. Some statements may not apply to every LGR.

1.1 Label Generation Rules (LGR)

A set of label generation rules for a zone governs the set of labels that may be allocated and eventually registered in a given Zone. A zone may support multiple LGRs if it caters to multiple languages or scripts. While an applied for label is always validated using a single LGR selected at application time, collisions would be determined by a Common LGR created for the zone as described below.

Logically, any LGR contains four parts:

1. the rules that define allowable Unicode code points (the repertoire),
2. any code point variants that can be substituted to form a variant label (the variant rules),
3. the disposition of any resulting label (whether it may be allocated, or is automatically blocked), and
4. a set of whole-label evaluation rules that determine whether the output of the previous three portions is still an acceptable label

(Adapted from [Procedure]).

The Label Generation Rules are expressed using a standard format defined in “Representing Label Generation Rulesets in XML” [RFC7940][1]. The XML format does not separate the LGR cleanly into the four logical parts described above, but it does provide for a mechanical computation of the status of any label as valid or invalid, and if a valid variant, as to whether that variant is allowed to be allocated, or is instead automatically blocked.

If a zone caters to multiple languages and scripts, with each of them using separate rules, a merged, Common LGR is needed to manage interaction of labels across scripts and languages (such as blocked cross-script variants).[2] The process of creating this Common LGR is called “integration”. That process is described in some detail in the overview for the Root Zone Label Generation Rules [RZ-LGR-Overview].

[1] The remainder of this document assumes that the reader is at least familiar with some of the general concepts presented in that RFC.

[2] Even where the individual LGRs contain all the cross-script or cross-reference variants, a merged LGR is required.

A merged, or Common LGR specific to the Second Level Reference LGRs has been computed for use with all Reference LGR not intended to be exclusively used in a standalone fashion. This Common LGR can be used to determine label conflicts among any combinations of its constituent LGRs as described in Section 3.1, "How to use the Common LGR in Label Processing". If a registry modifies any of the LGRs during adoption from the reference LGRs, recomputation of the Common LGR may be necessary before it can be used with the modified LGR.

1.2 Reference LGR Files

A reference LGR is a set of label generation rules that can be used as is or serve as the starting point for fine tuning an LGR for a specific combination of languages and scripts for a given zone. If a zone supports multiple languages that all share the same script, it may be simplest to just support the script LGR.

The normative definition of each reference LGR is provided as an XML file. The LGRs are expressed using a standard format defined in "Representing Label Generation Rulesets in XML" [RFC7940].

Each of the reference LGRs are provided for a specific language or script as indicated in the <language> element of the XML file, with script-specific LGRs using the "und-XXXX" form of language tag as specified in [RFC 7940]. For such LGRs, the language element generally identifies the predominant script, for example, characters from the "Common" script are included without an additional <language> element of "und-Zyyy". In a few cases, the single language tag applies to a mixture of scripts.

Each LGR contains all the specifications applicable to the labels from that language (or script), and only those. Each file contains a detailed human-readable description, a repertoire with optional variants, and context or WLE Rules, as well as detailed references[3] that link each included code point to a reference providing data for justifying its inclusion.

From each XML file, a non-normative HTML presentation is generated mechanically with additional formatting. These HTML files are provided for the convenience of the reader. The HTML presentation provides a formatted Description section, and is augmented by summary data as well as data extracted from the Unicode Character Database [UCD]. In addition, it provides interactive features such as links to code points, references and other items defined in the LGR.

The set of [Second-Level Reference] files can be found on this website:
https://www.icann.org/resources/pages/second-level-lgr-2015-06-21-en

1.2.1 LGRs Omitting Cross-Script Variant Listings

The majority of the LGRs are presented in a form intended to be easy to use as "standalone" LGRs, that is, in a zone supporting only a single language or script. These LGRs are formulated to not show the interaction that would arise if they are used in combination with other scripts or languages in the same zone: any applicable cross-script or cross-repertoire variant definitions are omitted. A small number of LGRs contain features that excludes their use in anything other than a standalone fashion; in a few cases, LGRs contain in-repertoire variants that make it possible to use the Common LGR to facilitate

[3] See the note in Section 7, "References", below.

concurrent use of the LGR with any of the other reference LGRs. These variants can be removed if not needed, but when removed, make the LGR fit for standalone use only.[4]

If used in a "standalone" fashion, all the required information for label processing is present in a single LGR file. In contrast, when using any of these LGRs in zones that use multiple LGRs in the same zone, additional information about cross-repertoire conflicts is needed. All the information for handling cross-script collisions between labels in such a zone is present in a single file, called the Common LGR.[5]

However, even for standalone LGRs, in particular the language LGRs that use the Cyrillic script, it is worth noting that many of their characters are obvious homoglyphs of Basic Latin (ASCII) letters while others are homoglyphs of extended Latin letters or letters from other European scripts.

The concurrent use of such LGRs with non-IDN labels (that is ASCII, or LDH labels), or with LGRs supporting the Latin script or a subset thereof in the same zone gives rise to the potential for "whole script confusables" (or even whole script homographs). Those are pairs of labels, one in ASCII (or Latin) and the other in Cyrillic in this case, which either have identical appearance, or are effectively indistinguishable when presented to the user. "Standalone" use for these LGRs is therefore not limited to IDNs, but would exclude non-IDN labels in the same zone.

There are some other pairs of scripts that give rise to whole script homographs. If concurrent use of multiple LGRs is anticipated, the cross-script variants defined in the Common LGR may be used to allow mutual blocking between such labels, as long as each of the LGRs is compatible with use of the Common LGR. (See in Section 3 "Use of Multiple Reference LGRs in the Same Zone" below for more information).[6]

Note that some standalone LGRs for a limited repertoire, such as a repertoire for a particular language, may lack those in-script variants that are irrelevant to users of that language. However, when a script is used without being limited to that language context, these variants may again be required. Conversely, LGRs focused on users of a specific language may include additional in-repertoire variants based on the usage in that language environment, even though they might not be present in a more "generic" LGR for the whole script. Using the Common LGR with such cases, any variants that do not correspond to variants in the Common LGR or vice versa variants would need to be disabled or added, respectively. Alternatively, the Common LGR would need to be modified to match. Otherwise, the results will be inconsistent. Generally, once multiple languages are to be supported concurrently in a given LGR, it is preferable to use the corresponding script LGR which properly mitigates the interactions.

1.2.2 LGRs Defined for Concurrent Use

Under the assumption that LGRs not limited to standalone use may be used in concert with other LGRs, a set of "full-variant" LGR has been defined that <u>collectively</u> contain the required cross-script and cross-repertoire variants needed to mitigate whole-script homograph labels. The variant mappings to other

[4] The decision to delegate labels based on features not compatible with the Common LGR is not reversible. Later use of the Common LGR may result in unresolved label conflicts.

[5] See Section 4.5.8 "Implicit Cross-script Variants" for more information on cross-script variants

[6] In limited cases, a standalone LGR deviates in the design of in-repertoire variants in a way that makes it incompatible with the Common LGR.

scripts may only be listed in one of the affected LGRs, but by symmetry and transitivity they apply to labels of all the other scripts affected: All cross-script variants between two scripts, apply equally to labels from either of them, no matter which LGR contains the definition. In some cases, in-repertoire variants will be imposed as side effect of variants with other scripts or languages. These are always identified.

Note: actual collision testing requires the complete, merged variant sets, containing all symmetric and transitive mappings. Creating these sets requires integrating all supported LGRs and using the resulting merged or “Common” LGR for all collision testing[7]. The use of the Common LGR in collision testing is described in Section 3.1 “How to use the Common LGR in Label Processing” below. (See also Root Zone LGR Overview [RZ-LGR-Overview] for a description of that process.)

As long as the set of “full-variant” LGRs provide all the variants necessary and are compatible with their standalone equivalents with respect to repertoire and other aspects of the LGR definition, it is not necessary to actually use the “full-variant” version of the LGR when doing label validation or any other part of the processing that doesn’t require the Common LGR. Where available, a standalone, but compatible LGR for the script may be used instead.

1.2.3 Labels Affected by Variants

Which specific label applications affected by blocked variants depends on which labels are already registered. Even if a substantial number of code points have variants, it does not automatically mean that a similar percentage of labels has variant labels, nor that any given label will be blocked.

Any label containing a “unique” code point, that is, one that does not have a variant, will itself be unique, and therefore not have a cross-script or cross-repertoire variant *label*. As measured by typical word lists tested against these LGRs, the percentage of unique labels can be very high, even for LGRs where significant numbers of code points have variants defined. In any case, the first instance of a label applied for from a given variant label set may be registered, even if subsequent labels are blocked. This essentially represents an extension of the usual “first-come-first-served” approach to registering labels.

1.2.4 Concurrent Use with non-IDN labels

Because non-IDN labels (so called LDH labels) never had variants to other LDH labels, no variants will be defined that would impose any variant relation *inside* the ASCII set. This makes it possible to retrofit LGRs built on these Reference LGRs into any zone that already supports LDH labels.[8] Some caveats apply.

Any delegated LDH labels in that zone will now have potential variants among the IDN labels, and those IDN labels would be blocked from being allocated. After the first IDN labels have been allocated, they may have LDH variant labels and thus block allocation of some future LDH labels in such a mixed-use zone. Common LGR

[7] While only the Common LGR is used for collision testing in that scenario, a key consequence of the “integration” technique used to create the Common LGR is that every cross-script variant **must** be defined in one or more of the contributing or “element” LGRs used in the integration process (for the Second Level Reference LGRs, the set of script-LGRs among the reference LGRs is used in the merge, see [Common]).

[8] Any confusability between LDH labels will have to be mitigated using other means.

The Common LGR contains the cumulative repertoire, WLE rules and all non-reflexive variant mappings (with type set to “blocked”) merged from the contributing LGRs. The Common LGR thus presents the complete data and specification needed for conflict checking with any existing label in a Zone using more than one of the Reference LGRs concurrently, independent of script. (This conflict checking proceeds by calculating and comparing “index labels”, see Section 3.1.1 “Steps in Processing a Label” below).

Note that the Common LGR *cannot* be used to determine the *validity* of a label because the validity of a label depends directly on the specific subset of the overall repertoire that is defined for a given script or language. (Simply applying the merged LGR would also result in returning mixed script labels as valid). The validity of a label may further depend in some circumstances on the script-specific definition of variants. For these reasons, the merged LGR is only used for collision testing of valid labels.

1.2.5 Reference LGRs Used as Input to Common LGR

The individual Reference LGRs and the merged file produced from them, the Common LGR, serve different purposes when processing a label. A registry adopts one or more LGRs based on individual Reference LGRs following the instructions in the reference LGR. These adopted LGRs become the actual LGRs deployed by the registry for a given zone.[9] At the time of registration, the applicant or the registrar system selects the script or language in the context of which the applied label is to be considered. That selection determines which LGR is used in processing the application.

Each of the individual LGRs presents the complete data and specification to determine the *validity* of a label as well as to *validate any proposed allocatable variants* for the label, when applied for under that script or language. However, they generally do not contain the complete set of variant definitions needed to detect cross-script or cross-repertoire collisions between labels.

As the per-script Reference LGRs (including those with full variant listings) were integrated (or merged) into the Common LGR, they are also called “Element” LGRs in the following. Strictly speaking, language-based LGRs, LGRs marked “standalone” or any reference LGRs modified during adoption are not element LGRs, but as long as they satisfy certain constraints, can be treated as described here for element LGRs for the purposes of label processing.

2 Use of Reference LGRs

These LGRs are intended as a *reference* for registries in defining the actual LGRs that correspond to the registry policies for a specific zone. The reference LGRs defined here can be adopted by a registry with unchanged technical contents. In that case, only minor administrative details need to be updated. Or they can be modified further before adoption. Some possible modifications have been anticipated here. (For example, see “Repertoire Extensions” below). Even if no modifications are made, there are some fields in the metadata, such as date and applicable zone as well as additional information like contact information that must be provided before submitting the LGR to the IANA Repository of IDN Practices [IANA]. Each reference LGR contains a set of instructions on how to modify the XML source file.

[9] The same adopted LGR may be used in several zones, as specified in the “scope” elements in the LGRs metadata.

2.1 Repertoires

The information in these reference LGRs represents the best available knowledge of the code points suitable for IDNs for users of a given language or script. As [RFC6912] makes clear, IDNs are intended to be *reasonable mnemonics*, and not for the faithful representation of any possible text in a given language. However, what is a reasonable mnemonic is informed by the language of the user community. Letters or diacritics that are unfamiliar in appearance do not make good mnemonics even if they are technically part of the same script. In that sense, where these LGRs have been developed for a given language that fact can also be understood as meaning that they were developed with the expectations of users of a particular language in mind. Code points that are part of a script, but are of uncommon or specialized use and therefore unfamiliar to general users of a script would likewise not make good candidates for reasonable mnemonics.

2.1.1 Subsetting the Repertoires

Creating a subset of one of these LGRs would generally represent a more conservative choice (see [RFC6912]). However, the final choice will always have to be made in tension between the two goals of usability and conservatism. There are several issues to consider when contemplating the creation of a subset. The first affects usability. For example, consider the case of reducing any Latin-based LGR to the letters "a-z". This is undoubtedly a conservative choice. But it also eliminates any gain in usability compared to non-IDN labels. A subset should always be a carefully designed consistent whole. (See also Section 4.2 and Section 4.6)

The next concern applies to LGRs that contain variants. For those LGRs, the effect of subsetting on the variant sets must be reviewed thoroughly. For each code point to be removed, all variant mappings related to that code point must also be removed. Once these are removed, it may be difficult to add the code point back again in a future version of the LGR. Adding a code point in a future version requires a thorough analysis of variants and WLE rules which may affect the existing registration and create the risk of stability issues. Finally, any rule that depends on the definition of a given code point must be updated if that code point is removed.

How to fully address these complications or those arising from having grandfathered certain registrations is beyond the scope of this document.

2.1.2 Overlapping Repertoires

Additional policies, variants and rules may be needed if any of these reference LGRs is adopted along with other LGRs that have an overlapping repertoire.[10] This is especially relevant in the case of LGRs defining variants or LGR-specific rules.

2.1.3 Repertoire Extensions

There are two ways of extending these LGRs. The first is by allowing additional code points that are considered widely used in the context of a given language or script, or that belong to languages in the region and show up in names or unassimilated loan words. Some of the reference LGRs directly provide information on a suitable set of such extended code points (see Section 4.2).

[10] An example of an overlapping repertoire is the shared use of the Han script for Japanese and Chinese.

The second is the use of a number of language-based reference LGRs as “building blocks” in assembling local, regional or script-based LGRs. When used in that fashion, care must be taken so that the resulting LGR provides for a consistent treatment of variants and WLE rules.

Any LGRs to be combined should be from a single script. The issue of mixed script labels is addressed in [RFC5890]. In combining LGRs into a single LGR it is recommended to first combine their core repertoires and, after eliminating duplicates, to consider possible additions from the extended sets separately. A combined LGR could have multiple “language” elements to indicate the range of languages covered, or a single language element indicating the script (see [RFC7940]).

For many scripts, the task of creating a script-wide reference LGR has been carried out based on the Root Zone LGR effort, but in some cases, regional LGRs might be of interest and would benefit from the construction process outlined above, or from subsetting a script LGR based on regional needs.

2.2 Variants and Rules

Some reference LGRs contain variants that are “enabled” by default in a way that can be disabled cleanly, if desired. If there is a desire to modify the variant set beyond that, care must be taken to avoid problematic interactions and other issues. For more information see [RZ-LGR-Overview].

When merging LGRs, for example to create a regional LGR, or when using LGRs with overlapping repertoires in the same zone, the “rules” element in the XML must be given special scrutiny. Some of the “rule” and “class” elements may be merged safely. Others may have to be renamed to keep them distinct. “Action” elements must be present in the order required for proper precedence in the merged XML.

That said, most of the LGRs presented here have a generic, default “rules” element. Any two LGRs with only the default rules can be merged and a single copy of the default rules appended.

Merging repertoires for the purpose of creating an extended LGR (for example a regional LGR supporting a number of languages, but not all uses of a given scripts) is a different process from mechanically merging LGRs to create an “integrated” LGR that can be used in detecting label collision. For more information on the process of integrating LGRs, see [RZ-LGR-Overview]).

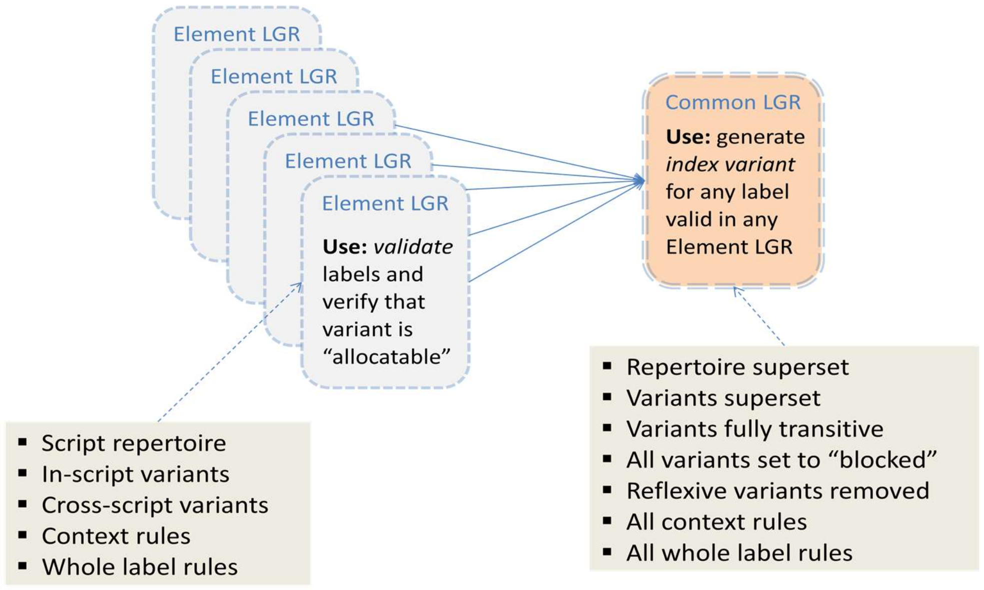
3 Use of Multiple Reference LGRs in the Same Zone

If a zone supports labels from multiple LGRs, cross-repertoire (or cross-script) variant labels may exist (see Section 6.3 “Cross-script variants” in [RZ-LGR-Overview] for a discussion). This situation arises when multiple LGRs are used, each defining the valid labels and variants for a given script or language (in contrast to the case of a combined repertoire as discussed in Section 2.1.3 “Repertoire Extensions” above. For efficient resolution of cross-repertoire variants, a special merged or “common” LGR needs to be created that is optimized for the task. For a discussion, see Section 5.2 “Common LGR” in [RZ-LGR-Overview]). As long as the common LGR file was created using all of the multiple LGRs that have cross-repertoire variants with each other, it can be used for that purpose – even if it contains information from additional LGRs.

Note that these reference LGRs are designed with the assumption that any native digits are variants of the corresponding ASCII digits by value, if both occur in the same LGR. When multiple LGRs are used in the same zone, transitivity requires that all such native digits become cross-script variants of each other. The effect of this transitivity is omitted from the files describing the individual LGRs; it is present in the common LGR to allow proper cross-repertoire collision detection. (See also, Section 4.5.2, "Digit Variants".)

A Common LGR file for a combination of these reference LGRs has been provided to aid in deployment of multiple LGRs for the same zone [Common]. The preamble to the file states which reference LGR files were included in its preparation. That file also contains notes on common design issues and conventions applicable to the reference LGRs. The file can be used as provided for any combination of LGRs that match some or all of the contributing LGRs or that are proper subsets of one of the contributing LGRs.[11]

Figure 1. Relation between Element and Common LGRs



3.1 How to use the Common LGR in Label Processing

3.1.1 Steps in Processing a Label

In order to determine the disposition of a proposed label, it is evaluated against the selected Element LGR and the Common LGR in the following steps (see also Figure 2, below). Note that the Reference LGRs implement all context restrictions defined in the IDNA 2008 protocol, including allowable locations for the HYPHEN-MINUS character. Therefore, LGR processing fully accounts for these constraints. However, the LGRs do not contain the information needed to normalize or lowercase the label, or to check for maximal allowable label length. These steps have to be performed prior to LGR processing.

[11] In the latter case, any in-script variants defined in the Common LGR will be imposed even if the actual LGR subsets the contributing LGR's in-script variant set. Any cross-script variants will be applicable as specified in the common LGR.

1. *Verify that a proposed label is valid by processing it with the Element LGR corresponding to the script or language that was selected for the label in the application.*

   This check will determine whether all code points in the label are defined in the LGR, and whether each code point meets all the context rules defined for it. In addition, all whole-label rules are evaluated; if a disposition other than "valid" results, the label is invalid.

   At this first step, do not enumerate all variants. However, as part of checking validity it is necessary to evaluate any *reflexive* variants[12], and apply dispositions based on their types. For example, if any reflexive variant is of type "out-of-repertoire-var", the label will be invalid.[13]

   For any invalid label, stop the processing and reject the proposed label.

2. *Process the now validated label against the Common LGR[14] to verify it does not collide with any existing delegated labels (or any of their variants, whether blocked or allocatable).*

   Each label and all its variants form a variant label set. For the Reference LGRs, all variant label relations are symmetric and transitive at the code point level, which means that all such variant sets are disjoint (do not overlap). The resulting sets of variant labels are also disjoint, but not all variant labels may be accessible from any other variant label.

   *For each label, calculate an Index Label identifying the set (the element lowest in code point order) as described in Section 3.2 "*Index Label Calculation*".*

3. *Determine whether the index label for the proposed label is unique.*

   Any two labels resulting in the same index label will collide: either with each other or with one of the variants of the other label. The Common LGR is defined to guarantee that all members of a variant label set produce the same index label. This step requires a list of all index labels for all labels already delegated. (See also Section 3.3.2 "Requirements for Index Labels")

   For any label that does not have a unique index label, stop the processing and reject the label. This label collides with existing labels and is "blocked".

If the proposed label is accepted, it is called the "original label" and can be processed for delegation according to registry policy. It can also be used in evaluating any proposed candidate variant labels.

[12] A *reflexive* variant is one that maps to itself.

[13] Some of the LGRs use reflexive variants to indicate a code point that is unmodified from the original code point (identity mapping). In these cases, the RZ-LGR guarantees that any valid label that is an identity variant of the original label returns a disposition of "valid".

[14] The Common LGR used in this step must have been created using the Element LGR used in step 1, or an LGR that uses a subset of the repertoire for the LGR that was used. More specifically, the LGR used in step 1 must not report a label as valid if it is not also valid under the LGR for the given script that was used for the Common LGR merge.

3.1.2 Steps in Processing a Proposed Variant Label

4. *Now that the original label is known to be valid, and not in collision, use the appropriate Element LGR to verify the validity any proposed candidate variant labels for that label.*

   This step proceeds in full analogy to Step (1) above.

   If the proposed candidate variant label is not a valid label under the Element LGR, stop the processing and reject the candidate variant.

5. *Verify the allocatable status of the proposed candidate variant using the appropriate Element LGR.*

   Verifying that a specific candidate is a variant and that that variant has the disposition of "allocatable" is a straightforward and computationally inexpensive process, which is the reason why the process assumes that the application would request specific candidate variants for verification, where applicable.

   In contrast, the enumeration of all potentially allocatable variants may be computationally expensive or even prohibitive. This is true even in cases where context rules and other constraints reduce the final number of allocatable variant labels: Some restrictions can only be applied after each candidate variant label has been enumerated; therefore, they still require the full permutation across all potential variants. For this reason, candidate variant labels should be identified by the applicant based on an understanding of the linguistics of the proposed label, instead of the registry attempting to provide an automatically generated "pick list".

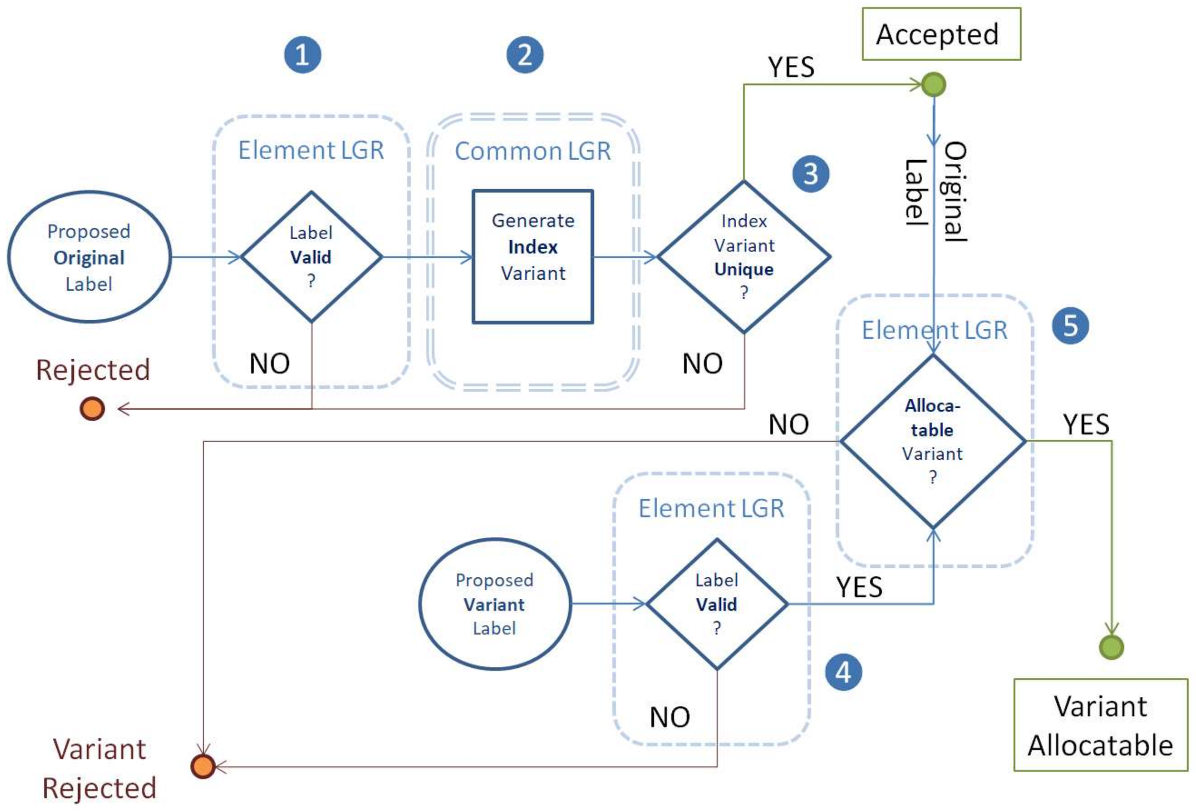
   Whether a label is an allocatable variant depends on the original label, the allocatable status is not symmetric. There may be multiple combinations of variant mappings applied to the code points of an applied for label that result in the same candidate variant. In this case, treat the candidate variant as allocatable, as soon as any mapping results in a variant label with disposition of "allocatable".

   If a candidate is neither a variant nor allocatable, stop the process and reject the proposed variant.

If the proposed variant is accepted, it is now "allocatable" and can be processed for delegation according to registry policy. Where an LGR defines multiple allocatable variants for a given label, registry policy may further restrict how many can be applied for.

A valid label and any verified allocatable variants constitute the result of the LGR processing and form the input into any subsequent stages of the application and registration process. The following figure shows a schematic overview of the steps in label processing.

Figure 2. Steps in Label Processing



3.2 Index Label Calculation

The most commonly defined variants are those that substitute single code points, where neither the code points nor the resulting labels are subject to code point context rules or whole-label rules. Where code point context rules or whole-label rules do apply, there may be potential issues in index label calculation that require careful attention when designing LGRs. In cases where *n:m* variants are defined (mapping code point sequences of length *n* to code point sequences of length *m*), additional complications may arise if some of the code point sequences share common code points, or are themselves part of other variant sets

In these and other cases discussed in Section 6 “Design Notes for the Root Zone LGR” in [RZ-LGR-Overview], a variant context rule may need to be defined on the variant so it is only defined in situations where the substitution is valid. Otherwise, the resulting sets of variant labels are either not transitive, or they may present difficulties in efficient computation of index labels, an essential tool to quickly compute collisions between variant labels.

For some combinations of cross-script and overlapped variants it may not be feasible to specify fully transitive code point sets across all LGRs. As long as variant label sets are disjoint (and have unique index labels), the listing of redundant sequences solely for the need of cross-script variants for an overlapped sequence can be avoided without effect on the integrated LGR. For an in-depth discussion of see Section 6.6 “Overlapped Variants” in [RZ-LGR-Overview].

3.2.1 Generating Index Labels

1. Index label generation starts with a *valid* label.
2. Index label generation proceeds beginning to end in code point sequence.
3. Any context rules for repertoire elements (code points or sequences) are ignored.
4. At each point, for any code point or sequence for which a variant is defined, the lowest variant in code point order is substituted (or the original code point or sequence retained if lowest or without variant).
5. If two variants are such that one is a prefix of another, the shorter variant (i.e. the prefix) is used.
6. In determining available variants, any variant that has a variant context rule and does not satisfy that rule is ignored.
7. Processing continues at the next code point after the code point or sequence that was the source of the variant mapping.
8. If more than one code point/code point sequence start at a given point, a separate index label candidate is calculated for each case and the processing continues for that candidate at the end of the given sequence.

   This case can arise, for example, if both a sequence and a leading part are separately defined as members of the repertoire. Each division of a label into sequences is called a *partition* and an index label candidate is produced for each possible partition of the given label.

9. At the end, the lowest index label candidate in code point order becomes the Index Label.

Whether or not an index label is a valid label does not matter. Therefore, index label generation ignores any code point context rules or whole-label rules.

Note that an index label may be a mixed script label. Because index labels are computed based on the Common LGR containing a merged repertoire, any "mixed script" labels are notionally in-repertoire and labels from different scripts can be tested against each other for collisions, even if their index label is a mixed-script label.

3.3 Defining Index Labels

In order to efficiently detect whether a label is blocked by another label, an index variant label (or index label for short) is computed for both. If the index labels are equal, the two labels are variants of each other. Assuming an existing list of index labels for all registered labels, an application for a new label can be very quickly checked for collisions, as long as the computation of the index label itself is efficient. To ensure efficient calculation under certain variant set definitions, it is important to be able to calculate the index label in a single pass (as described above) and still get a correct result.

By contrast, any calculation that requires enumerating all variant labels may well be prohibitive, as some longer labels may create very large numbers of blocked variant labels. Even in the case of allocatable variants, where additional rules or conditions limit their number, it may not be possible to perform an enumeration in the general case. Often, the full permutation of all putative labels has to be performed before those that are actually allocatable can be determined. Akin to the case of index labels, the solution is to turn the process around and present an original label together with a candidate variant label. Verifying that a candidate variant is valid and allocatable is computationally much cheaper.

3.3.1 Transitivity of Code Point Variant Sets and Variant Label Sets

Transitivity means that all variants in the set are variants of each other. See RFC 8228 for a discussion of this and other concepts related to variants.

For checking collisions, it should not be required to enumerate all blocked variants – doing so is prohibitive in terms of performance. Likewise enumerating all allocatable variant labels can be expensive in the general case, and is best replaced by verifying that a proposed candidate is an actual allocatable label for a given original label. Therefore, the only requirements are that an LGR be well behaved as far as index label calculation is concerned, and that all allocatable variant labels can be reached from a given original label.[15]

When code point variant sets are defined for code point sequences in LGRs where subsequences of the same sequences are part of the LGR's repertoire (and especially, if they have variants in their own right) then a *variant label set* may not be transitive, or non-overlapping, even if each *code point variant* set is defined in a formally transitive manner.

Any LGR with such overlapping sequences requires special attention to ensure that it is well behaved.

3.3.2 Requirements for Index Labels

For the index label method to work, the space of all labels and their variant labels must be divisible into variant label sets so that

- any label and all its variants belong to the same set
- no two sets overlap
- all labels in the set generate the same index label

If these conditions are met, two labels with the same index label are members of the same set, even if one is not a directly accessible variant of the other.

For these requirements, it is inessential whether any enumerated variants are also valid labels or not, as long as any invalid variant labels also belong to only one set.

3.3.3 Impact on Reference LGR Design

For many complex scripts, code point context rules and whole-label rules restrict the set of valid labels. If putative labels are first evaluated against the element LGR to make sure that they would be valid, and then checked against the common (merged) LGR for collisions (as recommended above in Section 3.1.1, “Steps in Processing a Label” above), it is not necessary to ensure that invalid labels are well behaved under index label calculation.

[15] The “allocatable” status of a variant mapping between two labels is not symmetric or transitive; it depends on which one is the originating label.

In verifying that proposed variant definitions were well-behaved[16] for valid labels, it was found that there was a dependency on the choice of index label: for the Reference LGRs, the variant definitions are only well behaved under the assumption that the index label is calculated as described here, using the lowest variant code point value. Theoretically, an index label could just as well have been calculated using the largest variant, but doing so would require changing or adding some variant definitions.

Therefore, the Reference LGRs now treats the Index Label Calculation presented in Section3.2 "Index Label Calculation" above as a requirement.

4 Contents and Design Notes

The development and review of these reference LGRs, including future updates, proceeds according to the [Guidelines]. The reader of this document is assumed to be familiar with the [Guidelines]. The following notes provide some additional highlights as well as information not specific to any individual reference LGR.

4.1 Repertoire

The repertoire for each reference LGR is either based on the Root Zone LGR with suitable additions for the second level, or based on a consensus repertoire derived from the sources consulted and listed in the LGR. In the case of many of the language-based LGRs, the repertoire caters to more than the code points minimally needed to write the native vocabulary of the language: it also includes code points that are in common use for loan words and the like. Where a language has multiple user communities with some variation of usage, a single, combined LGR is produced. The details are described in each of the LGRs. For some languages or scripts, the repertoire may also explicitly list certain code point sequences.

4.1.1 Root Zone LGR and Second Level Additions

Many of the reference LGRs are directly derived from the relevant Root Zone LGRs. With few exceptions, these LGRs are developed on a per script basis and they deliberately limit their repertoire to code points in everyday common use, eschewing historic, obsolete or special purpose code points in limited use. Generally, though, they tend to support all languages in active, everyday use written in a given script.[17]

In some cases, a language-specific reference LGR has been derived by starting with a script-base Root Zone LGR and limiting the repertoire to code points used in a specific language, utilizing the language-specific source information provided available in the Root Zone LGR and additional input from other sources as documented in the LGR.

[16] Well-behaved in this context means that any two (valid) labels that are variants of each other do not lead to two different index label. In some instances, two valid labels that lead to the same index label may not have a direct variant relation or not a symmetric & transitive one. This can arise in cases where the mapping "should" exist, but where its formal definition would require added and unnecessary complexity.

[17] Not all languages that may be written in a given script use that script for common, everyday use. Also, some languages are primarily spoken languages only, or they are otherwise in very limited use. Such languages are generally ruled out as justification for adding characters or features to an LGR and are only incidentally supported.

In all cases there have been additions to the repertoire specific to the second level. Generally, these include the HYPHEN-MINUS and either the common (ASCII) digits, or the script-specific digits for that script, or both. The selection of digits was based on research into the common, everyday practice of actual digit use for the various scripts. In isolated cases, some commonly used characters ineligible for the Root Zone were added as well. For CJK scripts, the ASCII letters are also included to match standard practice for the Second Level.

For some scripts, CONTEXTO code points (like '·', used in "l·l") or CONTEXTJ code points (like ZWJ and ZWNJ) are used in an integral way in some orthographies to mark distinctions between words, as opposed to merely stylistic use. It seemed possible and desirable to support them with strict limits and proper safeguards in a second level LGR (for example by enumerating specific allowable combinations, and/or setting context rules). All such additions generally proceeded in consultation with experts from the same panels that developed the Root Zone LGRs.

4.1.2 Sources for Repertoire other than the Root Zone

In determining the repertoire for those language-based LGRs that were not derived from the Root Zone LGR, a large number of sources was investigated, from spelling dictionaries released by language authorities, applicable RFCs and national or international standards, to sources such as commercial or online dictionaries[18], the Common Locale Data Repository [CLDR] (a project of the Unicode Consortium) and finally existing IDN practice for ccTLDs or gTLDs aimed at users of a particular language. The sources and their contribution to the development of the repertoire are documented in detail in each of the LGRs.

The repertoire for any such language-based reference LGR, though derived independently and from different sources, is typically a subset of the *repertoire* of the reference LGR for the corresponding script.

In some cases, there has been direct community input during public comment proceedings that substantively redefined the design of the reference LGR.

4.2 Extended Code Points

Many, though not all, of the languages supported by language-based reference LGRs are written by compact communities that are in contact with other languages in the same region or in the same country. In those cases, native users may have familiarity with or need for access to an extended set of code points, for example for names of people or places. Certain of the reference language LGRs provide for those code points by listing them as "extended-cp" if they are not already catered for in the core repertoire. As written, the LGRs treat these extended code points as ineligible for a label, but registries could easily remove the restriction to tailor such reference LGR to their needs before adopting the LGR. (See also Section 4.6.2, "Reference LGR-specific Rules").

This is in contrast to script-based reference LGRs that typically provide for the full repertoire needed for all languages sharing a common script, or those country-based LGRs in existing registries, that provide

[18] Not limited to those issued by language authorities.

for the needs of users from the same country or territory, irrespective of whether they write a majority or minority language prevalent in the country.

4.3 Excluded Code Points

For most languages or scripts, there are among the consulted sources some that include a number of code points that are very rarely used, or that are historic or limited to special purposes, like poetry and religious works. Such uses are rarely germane to IDNs, and if included, could confuse users not familiar with them.[19]

Consequently, such code points have been excluded from these reference LGRs and documented as such; either explicitly in the LGR, or implicitly in the background documentation for the cited Root Zone LGR development (see, for example, code points excluded from the [MSR]). In contrast to the "extended" code points, which are specifically called out as likely targets for customization, excluded code points can be safely left unsupported.

If it is felt that some excluded code points (or any other code point not included) might be required, the reason could well be that the scope of the LGR no longer fits the original design, but has morphed from, for example, a language-based LGR to a regional or multilingual LGR. That would be an indication that the chosen reference LGR doesn't really apply and a better approach might be to support the whole script, or cut down a script LGR to size for regional use.

4.4 Sequences and Context Constraints on Repertoire Elements

Certain code points tend to occur only in fixed combinations. The repertoire contains such code points only as part of an explicitly specified code point sequence, or occasionally via restriction by context rule. This prevents unneeded combinations and primarily applies to combining marks such as diacritics (but also to certain clusters in complex scripts). Another important case is the use of invisible joiner or non-joiner characters the use of which should be tightly controlled to specific instances where their use is required (and expected) in the writing system as used for everyday communication.

Rendering systems may fail to provide a predictable presentation of combining marks if they are present outside of expected contexts, whether applied to unexpected base characters or, for example, repeatedly applied. In the latter case, in particular, there is a danger of "overprinting", which would mask the presence of an extraneous diacritic. Finally, some diacritics are easily confused with others. Allowing unrestricted combinations would allow these diacritics to be applied to base characters that normally take different diacritics, greatly adding to the risk of creating confusable labels.

Join controls that are present outside their expected context might be ignored in rendering, in the worst case, and because they are invisible by default, would result in two labels that are indistinguishable yet contain different code point sequences.

[19] Faced with an unfamiliar character, users may not recognize it as distinct from some character they know. Especially, if the unfamiliar character is present where a similar-looking familiar character is expected, the substitution may not be noticed or questioned.

Additional constraints are provided by Whole Label Evaluation and Context Rules (see Section 4.6). Whole label rules apply to the entire label, while context rules are evaluated at a given code point offset in the label. Both code points and code point sequences may have context rules.

Where both sequences and their constituent elements coexist, all valid partitions of the label into repertoire elements are evaluated for variants and disposition. Only the context rules for the members of a given partition are considered; a context restriction on a defined sequence does not apply to a partition where the constituent code points are taken individually and vice versa. Sequences can thus combine code points in ways otherwise prohibited by context rules.

4.5 Variants

A *variant label* can be defined in cases where two labels are indistinguishable, or where for reasons of the writing system, users are prepared to accept one for the other without noticing the substitution. They are implemented in LGRs by means of defining *code point variants*.

Before RFC 7940 provided a generalized method for expressing variants, the use of variants was limited to a few scripts for which specific RFCs provided both the notation and script-specific rationale.

Since then, and especially during the development of the Root Zone LGR, a more generalized understanding of the usefulness of variants has emerged. In this understanding “blocked” variants perform the first line of defense against duplicate registration of labels that may be indistinguishable to users, or otherwise readily accepted as “same” or as a valid substitute. Any collision between labels based on blocked variants can be efficiently detected and handled without manual attention.

Blocked variants are particularly useful in addressing security concerns for zones that support multiple scripts, where certain labels in one script can look like an ASCII label or an IDN label in another script.[20] However, their use is not limited to cross-script, or cross-repertoire scenarios, and there are many examples of in-script variants implemented as blocked variants.

Blocked variant labels are mutually exclusive, but that exclusion would also apply to dual registration by the same entity, which might otherwise be desirable. Judiciously used, “allocatable” variants provide a safety valve, by allowing a single entity to register two (or more) labels that would otherwise conflict. This may be appropriate whenever two labels aren’t visually identical, but represent two orthographically significant spellings that users may not treat as different; where one of the forms is a widely accepted fallback; or where each form is confined to a specific sub-group of users[21]. By their nature, there are no cross-script or cross-repertoire variants that are allocatable.

[20] Different concepts of “sameness” or substitution may exist simultaneously for some code points. It is sometimes not possible to satisfy all of them simultaneously in an LGR due to the requirement that all variants must be transitive. If care is not taken, the result could be that two code points that are clearly distinct are treated as variants of each other, because each individually is a variant of the same code point, but for different reasons.

[21] This most commonly refers to systematic letter substitutions, like traditional *vs.* simplified Han ideographs, and not so much to cases like “color” *vs.* “colour” that are not systematic and have exceptions, such as “or” *vs.* “our”.

Code point variants defined in an LGR must have symmetric and transitive mappings[22]. In a few cases, this transitivity can lead to an imposition of in-repertoire variants for one LGR as result of cross-script variants with one or more other LGRs. For example, the Greek LGR, by long-standing practice, treats 'ι' and 'ϊ' as variants, while the Root Zone LGR work identified the need to map these two to Latin letters 'i' and 'ï' respectively (the Latin and Greek version of the letters are not rendered distinct in all fonts). As a result of the required transitivity, the two Latin letters would pick up an in-script variant mapping in any zone where Greek is also supported.

Now, for an LGR targeting an English-language repertoire beyond pure ASCII, not allowing a registration of both the labels "naive" and "naïve" to some unrelated entities can be seen as a win for security with limited downside. However, in some other contexts, such imposed in-script variants may limit the availability of some labels as soon as such a "variant" by transitivity has been registered.[23]

In the example given, the two forms both exist, and the fallback (without diaeresis) is readily accepted by all users of English. However, in cases where one or the other two forms is systematically preferred by different communities, it may be advisable to allow one of the labels as an "allocatable" variant, so that a web resource can be simultaneously offered to both communities with their preferred form.

The reference LGR for the Latin script, for example, following the Root Zone LGR from which it is derived, contains all the variant mappings needed for integration with other scripts, particularly Greek and Cyrillic, but also others. On the other hand, the various language-based LGRs that use the Latin script are not configured to be used with other LGRs or other scripts in the same zone. They are standalone LGRs that do not contain variant definitions for interoperability. If it is desired to support multiple languages using the Latin script, it is preferable to use the reference LGR for the Latin script as a starting point.

Finally, it should be noted that not all kinds and degrees of label confusion can be handled with variants. For some, other methods for detecting and mitigating confusable labels must be used.

4.5.1 Variants Defined

Several of the language-based LGRs do not include the definition of any variants, but most of the script-based LGRs do. Where variants are included, their selection is informed by existing registry practice, as well as by the work performed at ICANN on the script LGRs for the Root Zone. Any cross-repertoire (or cross-script) variants identified in the Root Zone have been retained here for use in zones that support reference LGRs for more than one script or language. (See Section 3 "Use of Multiple Reference LGRs in the Same Zone").

LGRs intended for specific languages may not require the same tradeoffs as script-based LGRs catering to a wide mix of languages. Some of these LGRs may reflect language-specific practices that would be

[22] This statement refers to the mappings, but not to the variant type, such as "allocatable" vs. "blocked".
[23] The restriction imposed by such in-script variants should not be overstated. In the context of the whole label, they apply only if every other code point is either identical or also a variant. The presence of even a single code point position at which there is distinct (non-variant) code point in each of the labels will render the labels distinct.

hard to generalize across a script. This may result in their in-repertoire variants deviating from the corresponding script LGR.

4.5.2 Digit Variants

All the LGRs support the common (ASCII) digits. Any script-specific (or native) digits are treated as semantic variants[24] of the corresponding common digits. In zones where multiple scripts are present, all digit sets would become semantic variants of each other as required by transitivity.

In a few cases, different sets of native digits across scripts share forms that suggest the need to include variants based on homoglyph relations. There are cases where digits of different numerical value are homoglyphs of each other and defining them as cross-script variants would create potential conflict with variants based on numeric equivalence. (These are not defined as variants in the reference LGRs).

Some scripts share a single shape for use as both letter and digit; this will make them in-script variants of each other. As a consequence of transitivity, any letters affected will become cross-script variants of all corresponding digits in the other scripts (by value). However, the common digit 0 is not treated as a variant of letter 'o' – because that variant relation does not exist in LDH (non-IDN) labels. [25]

If a registry were to modify an LGR adding support for the native digits for any additional script, some in-script variant relations might need to be added for those scripts. In zones where multiple scripts are present, it might be difficult to accommodate both the required in-script homoglyph variants as well as variants based on numeric equivalence; thus, extensions to the sets of supported digits need to proceed with extreme caution and deliberation.

Finally, some native digits are homoglyphs of or are highly confusable with other code points (letters or digits) outside the script.[26] It is generally not possible to satisfy both the semantic variant relation and such cross-script homoglyph variants in a consistent way, while also maintaining transitivity. As a result, these reference LGRs prioritize the semantic variant relationships among digits, because allowing two sets of digits immediately creates an in-script issue requiring semantic variants. On the other hand, not all second level LGRs are necessarily in zones that support multiple scripts, making that need slightly less immediate. In consequence, any need for mutual exclusion of labels based on confusable digit shapes (homoglyphs) across repertoires would need to be addressed outside the reference LGR, for example via additional registry policies.

4.5.3 Variant Dispositions

All LGRs with variants support the disposition "blocked", meaning that either a label or its variant may be allocated, but never both. Some LGRs support "allocatable" variants, meaning that both the label or

[24] A semantic variant indicates a code point that has the "same meaning". For digits, this is taken as having the same value. For example, "123" and "١٢٣" both represent the same value and to users of Arabic may be interpreted the same, ignoring the choice of digit set as irrelevant.
[25] For the Myanmar LGR, this required implementing a special restriction to sidestep the issue as documented in the LGR.
[26] In contrast, where scripts share shapes between some of their own letters and digits, variants should be defined, even if that means some letters become variants of unrelated digits by transitivity.

the variant or both may be allocated to the same registrant. A few LGRs contain additional dispositions, such as “activated”, which implies that a label *should* be allocated. There are “optional” forms of some variant types defined that allow for simple customization of variant dispositions as described in the particular LGR. (In all LGRs, the original label, if not blocked or invalid, is reported as “valid”.)

4.5.4 Multiple Allocatable variants

The Chinese LGR resolves variant *labels* into “allocatable” and “blocked” only, however it utilizes a rather specific form of *code point* variant subtyping in an attempt to keep the number of allocatable variant labels manageable. The details are described in the LGR and references cited therein. (Some other LGRs use similar, but simpler forms of variant subtyping, for example, Greek and Latin).

Several LGRs use whole label evaluation rules to limit the available variant labels to those that consistently use either the one or the other variant for a code point throughout the label, disallowing mixed labels. In some cases, this is implemented by “no-mix” rules for the specific variants, in other cases there are rules that require that all variants to be in a single sub-repertoire (for example, a repertoire for a specific language). Occasionally, both of these approaches are used.

Finally, some LGRs support single “fallback” variants as allocatable variants from any of a number of permutations of the label using one or more instances of some special characters. (See Section 4.5.5 “Allocatable Fallback Variants” below).

Because the DNS does not natively support variant labels there is a cost to having multiple variants delegated, and thus a need to add such mitigation. The mitigation approaches in a particular reference LGR are not always sufficient on their own because it may not be possible to write generalized rules preferring some variants to others. Thus, registries should implement additional policies to limit the number of variant labels actually delegated.

Please note that even where typical labels may not generate an inordinate number of labels, it is possible to create pathological labels for some LGRs for which the theoretical number of allocatable variants could cause enumeration of allocatable variants to fail by exhausting resources. This can be true even in LGRs that use rules to invalidate all but a few of the possible allocatable variants. Therefore, it is <u>not possible to guarantee</u> that it is always possible to list all theoretically available allocatable variants for a particular label within practical limits.

Instead of enumerating variants, application processes should be designed to validate a single, proposed variant at a time.

4.5.5 Allocatable Fallback Variants

A *fallback variant label* is a single allocatable variant label that uses substitute code points or sequences for code points or sequences not available (or not allowed) in some contexts, but that would be required for a linguistically accurate rendering of some label. A fallback may not look like the intended label, but is generally recognized as an established “poor man’s substitute” for that label.

Widely encountered examples of fallback variants involve the use of ASCII equivalents for Latin letters or sequences. For example, the middle dot in Catalan “l·l” cannot be rendered in ASCII, but the sequence “l-l” can be used as a fallback. While it uses an incorrect representation of a key spelling distinction, for a domain name it may be a more important consideration, that users without access to a Catalan keyboard are usually able to type the “l-l” sequence instead.

Other scripts and languages have similar examples. For example, for the Sinhala LGR, all sequences containing a Zero Width Joiner (ZWJ) have a fallback variant without the ZWJ.

Common to all fallback variants is the desire to allow both the registration of a “preferred” or “intended” form of a label and a *single* fallback that avoids any use of special characters.[27] To support this, the variant mapping from the standard letter or sequence of is of type “fallback”, while the mapping in the other direction is of type “blocked”.

To complete the scheme, a third variant type “r-original” is used as a reflexive variant to identify code points that have a fallback mapping, but that appear in their non-fallback form in the original label, and thus “map to themselves”. A set of default actions is defined for use with all second level reference LGRs. These actions resolve as “allocatable” any label where all variants are of type “fallback”, and as “valid” any label where all variants are of type “r-original”. Any variant with a mix of variant type resolves as “blocked”.

With this system, registrants are able to apply for one label using their chosen form, no matter how many instances or permutations of non-fallback and fallback characters it contains, plus a single fallback variant containing no instance of any non-fallback character or sequence. Any other labels, such as those containing some other mix of non-fallback and fallbacks for the same label, would be blocked variants. As a result, fallback variants are limited to a single allocatable variant label, but also block any other variations of the same label beyond the one selected by the applicant.

Note that if the fallback itself is applied for as the intended label, no other label is allocatable and all other variant spellings containing non-fallback characters are blocked.

4.5.6 Multiple Blocked Variants

There is no limit to the number of blocked variants a label may produce under any of these reference LGRs. An exhaustive enumeration may be computationally infeasible for many labels. It is also not required for detecting collisions between labels that are variants of each other. Such testing can be performed efficiently by computing a single “index label” followed by a comparison of these index labels. The performance of this operation does not depend on the theoretical number of blocked variants. (For a more detailed description, and for additional notes how to ensure that index label

[27] Note that a “fallback” representation may spell a perfectly acceptable word in its own right, or be an alternative preferred spelling for the same word in a different locale. For example, German written in Switzerland always uses the sequence “ss”, while German written in Germany alternates between “ss” and “ß” depending on the spelling of a given word. Nevertheless, for users in Germany, “ss” is both a commonly accepted fallback for “ß” in contexts where the latter is not available, as well as the preferred spelling in words where an “ß” should not be used.

calculation is well-behaved, see [RZ-LGR-Overview], but also see Section 3.2 “Index Label Calculation” above.

4.5.7 Context Rules for Variants

Some variants require context rules to be well-behaved. See RFC 8228 or the discussion of this issue in Section 6, “Design Notes for the Root Zone LGR” in [RZ-LGR-Overview]. Any such context rules from the corresponding Root Zone LGRs have been retained, and a few additional ones have been added where needed by new repertoire (for an example, see the Devanagari LGR’s treatment of the HYPHEN).

Context rules for variants are also used in these reference LGRs to mark variants optionally as enabled. The context rule “enabled” matches any label (it always succeeds). Any variant marked with when=“enabled” is in force, while marking it with not-when=“enabled” causes the variant definition to be ignored in processing.

This scheme allows for simple adjustments as long as all variant mappings to and from the same code point are either enabled or disabled at the same time. (Due to a limitation of a single context at a time this scheme cannot be applied to make optional any variants that already carry other types of context restrictions).

4.5.8 Implicit Cross-script Variants

The listing of variants in the Reference LGRs may omit some cross script variants. Once defined in at least one LGR, they are incorporated into the merged common LGR where they are used for calculating index label used to check labels for collision, both in-script and across scripts. For this type of implicit variant, the merged LGR contains a full listing of the variant set under transitive closure, but not all of the reference LGRs repeat the mapping. Most reference LGRs only list their in-script variants.

A second form of implicit variant exists when a code point sequence has both an explicitly defined variant, but also a cross-script variant computed from variants of its constituent elements. For example, the Latin sequence “ss” has an in-script variant “ß” (sharp s or *eszett*) as well as a cross-script variant to the Greek beta (β). However, the code point “s” has several cross-script variants, for example to the Cyrillic letter Es “ѕ”. That means that the Latin sequence “ss” also has an implicit variant to Cyrillic “ѕѕ”, but by transitivity, the Cyrillic “ѕѕ” implicitly has “ß” and “β” as its variants. Note that Cyrillic “ѕѕ” as a sequence is redundant: it is not a target of an in-script variant, nor does it add to the space of available labels, as the use of two single “ѕ” in a row is already allowed.

For this second type of implicit variant, mappings involving sequences from other scripts are elided if they don’t contribute to the index label calculation. The Cyrillic LGR, having a mapping from “ѕ” to “s”, does not need a redundant sequence definition plus mapping from “ѕѕ” to Latin “ss” to make index label calculation possible. And a mapping from “ѕѕ” to “β” would likewise not contribute, because the Greek

LGR provides the mapping from “β” to Latin “ss”, allowing both a Greek and a Cyrillic label to map to the same index label.[28]

This elision is not possible if variants are allocatable in one or both directions, or if sequences are not redundant (for example if they contain unique code points, override context restrictions between their constituent code points, or where the constituent code points don’t produce the same index label).

4.6 Whole Label Evaluation (WLE) and Context Rules

WLE and context rules implement a further constraint on valid labels, for example, by limiting certain code points from occurring at the beginning of a label or occurring repeatedly or simultaneously with other code points in the same label. Context rules are a specialized form of a WLE rule that defines a constraint on the surrounding context for a given code point at that position in a given label (see [RFC7940]).

Some code points may be restricted not by context rules of their own, but due to the combined effect of context rules on all the other code points. Such code points are said to have “implicit” context restrictions. Finally, any code point sequence defined will implicitly override any restrictions resulting from the application of context rules for its constituting elements. Both WLE rules and context rules defined for the sequence as a whole remain unaffected, as would any implicit context restrictions evaluated for the neighboring code points.

4.6.1 Protocol-defined Rules

Because the XML format for the LGR supports machine-evaluation of labels for validity, these reference LGRs include all relevant constraints on labels defined in the IDNA protocol itself, other than normalization. In this way, the LGRs can be used to validate all constraints on any normalized label in one pass. For the purpose of these reference LGRs, an additional rule preventing the mixing of any two digit sets in the same LGR has been adopted project-wide.[29]

Common rules:

- Hyphen Restrictions — restricts the allowable placement of U+002D (-) HYPHEN (no leading/ending hyphen and no hyphen in 3-4 position). These constraints are described in section 4.2.3.1 of [RFC5891].[30]
- Leading Combining Marks — restricts the allowable placement for combining marks (no leading combining mark). This constraint is described in section 4.2.3.2 of [RFC5891].
- Digit Mixing — all LGRs support a rule to prevent mixing of multiple sets of digits in the same label. This generalizes the constraints found in [RFC5893].

[28] Because of the possible interaction between visual, semantic or fallback variants, an implicit variant, as in this example, can result in a blocked variant between seemingly unrelated labels, without this connection being immediately apparent in the LGR documents due to the way their presentation is streamlined.
[29] The names shown for these rules are specific to this project. Also, their spelling in the XML source may differ.
[30] The actual rules adopted here for the Hyphen-Minus are somewhat more restrictive, in particular it is prohibited in contexts where Katakana Middle Dot is allowed.

Rules for Right-to-Left labels:

- Leading Digit — restricts the allowable placement of digits for right-to-left labels (no leading digit in RTL label). This constraint is described in section 2.1 of [RFC5893].
- Mixed Digits — prevents the mixing of European and Arabic (Indic) digits. This constraint is described in appendix A.8 and A.9 of [RFC5893]. (In these reference LGRs this is a subset of the general digit mixing restriction.)

Context rules:

- Join Context — restricts the use of ZERO WIDTH JOINER (ZWJ) to consonant-conjunct context Indic scripts (immediately following a virama), to control required display of such conjuncts. This rule is described in Appendix A.2 of [RFC5892]. (This rule is implicitly satisfied by enumeration of all sequences that may contain a ZWJ in any reference LGR.)
- Surrounded by L — restricts the occurrence of MIDDLE DOT to be between two small letters L only, to permit the Catalan Ela Geminada (L-dot-L). This rule is described in Appendix A.3 of [RFC5892].[31]
- Japanese in Label — restricts the occurrence of KATAKANA MIDDLE DOT to labels containing at least one code point from any of these scripts: Han, Hiragana, or Katakana. This rule is described in Appendix A.7 of [RFC5892].[32]

Whole label rules:

- No ASCII only Label —restricts IDN labels to those having at least one non-ASCII code point. {RFC 5891]. [33]

For these reference LGRs, the protocol-derived rules, other than the common rules, have only been included if they are needed for labels that are valid in the given LGR. The description section of each LGR file lists the rules and their associated references.[34]

4.6.2 Reference LGR-specific Rules

A number of the LGRs contain additional LGR-specific WLE rules, reflecting further constraints on possible labels based on the nature of the language or script. These are documented in detail in the description section of the respective LGRs. These rules are generally derived from the Root Zone LGRs with suitable extensions in case of added repertoire elements (for example, see the Thai language LGR for the addition of a Thai abbreviation mark with Thai-specific rules).

[31] The actual rules adopted here for the Middle Dot are somewhat more restrictive by preventing an overlap between any two Ela Geminada sequences as, for example in “l·l·l”.

[32] The actual rules adopted here for the Katakana Middle Dot are somewhat more restrictive.

[33] By community request, this rule is not enforced in these reference LGRs, meaning that they can be used to apply for whatever subset of LDH labels the LGR may permit (most commonly ASCII digits plus Hyphen).

[34] For information on additional rules defined specifically for a given reference LGRs, see the description in the individual LGR.

The majority of these rules take the form of context rules on a given code point or sequence. (Sequences may be defined for the purposes of selectively overriding the constraints defined for single code points.)

In the case of many complex scripts, readers and layout engines expect the label to be series of valid syllables according to the rules of the writing system. Even though Unicode may encode the individual components of such syllables, arbitrary sequences of code points are not only unexpected, but can lead to security and predictability issues for identifiers. This is in contrast to alphabetic or ideographic scripts where the letter or ideograph is considered the unit, and arbitrary sequences are being used in identifiers. For that reason, complex script LGRs have WLE or context rules enforcing the deep syllabic structure of the writing system, while generally not attempting to enforce "spelling rules". While it may thus not be possible to make labels using non-existing syllables, there are generally no restrictions against creating "nonsense" words from well-formed syllables.

4.6.3 Special Rule for Optional Repertoire Items

Finally, some of the second level reference LGRs use a special context rule to support adapting a reference LGR to a specific zone. (For details, see Section 4.2 "Extended Code Points").

Special rule present in some reference LGRs:

- **extended-cp** —this context rule always fails. That means, as published, the LGRs do not allow the code points identified as extended by having been given a context of "extended-cp".

Simply changing that rule so it always matches would enable the entire set of extended code points without the need to edit the list of characters. Alternatively, the context condition could be removed from individual code points, or the "when" condition changed to its opposite "not-when" thus enabling them one by one. Likewise, to create a subset, a code point can be disabled by adding the "extended-cp" context condition. Doing so would mark the code point as deliberately not included instead of merely omitted.

Special context rule typically applied to variants in some reference LGRs:

- **enabled** — this context rule always succeeds. This rule matches any label, and therefore always succeeds. As published the LGRs, the specified variants with this context are unconditionally processed.

See the discussion of optionally enabled variants in Section 4.5.7, "Context Rules for Variants" above.

4.7 Metadata (and updates required in adopting a reference LGR)

The XML file format defines a number of elements for metadata. Several elements are not relevant to reference LGRs, but would be relevant to actual, deployed LGRs. These elements include the <scope> element defining the zone to which the LGR applies, or the elements giving the <validity-start>, and <validity-end> dates. In adopting a reference LGR as the LGR for a specific zone, values for these elements should be supplied. For more details, see [RFC7940]. Other required information may not have a dedicated XML element. In that case, the information should be added in the <description> element.

Others, such as the <date> and <version> fields are relative the Reference LGR document. Upon adoption by a registry as LGR for a particular zone, the <date> should be updated, the version numbering should be reset to 1 and the word “reference” removed from the optional comment in the <version> element.[35]

5 Review

These reference LGRs have been reviewed by the community as part of a public review process. Certain of the reference LGRs were additionally reviewed by members of the corresponding Root Zone Generation Panels or by independent reviewers with expertise in Unicode and linguistics, as well as IDNA and DNS security. Any LGRs were updated to reflect the input from the review. For the independent expert reviewers, review results, where available, are found at [Second-Level Reference].

5.1 Versioning

Each reference LGR has a version number and a date of publication. Any changes or corrections to a published reference LGR result in a new version number and publication date. The nature of any corrections or changes is documented in the description. Drafts and revisions published for public comment carry the same version number as the version targeted for publication, but not the final publication date. When a registry adopts the LGR for the first time, the version should be reset to 1.

6 Contributors

The following cumulatively lists contributors to the development of any Second Level reference LGRs.

1. Developers

Asmus Freytag
Michel Suignard

2. Expert Reviewers

Michael Everson
Nicholas Ostler
Lu Qin
Wil Tan

3. ICANN Staff

Sarmad Hussain
Pitinan Kooarmornpatana
Anand Mishra
Yin May Oo

4. Root Zone Generation Panel Experts

[35] Once a reference LGR has been adopted by a registry for a particular zone, even if not modified in substance, it is no longer considered a “reference” LGR and should not be identified as such.

An early version of all script-specific and some language-specific LGRs was reviewed by experts from the Root Zone LGR project. For a list of members for each Generation Panel, see the proposal document for the corresponding Root Zone LGR.

7 References

Not all online resources cited as reference in this overview or the individual LGR files have a stable URL. For resources not maintained by IANA, ICANN, IETF or The Unicode Consortium, a provided date of access may assist in locating an archived version of the resource in the internet archive.

[CLDR] CLDR - Unicode Common Locale Data Repository: http://cldr.unicode.org

[Guidelines] Internet Corporation for Assigned Names and Numbers, “Guidelines for Developing Reference LGRs for the Second Level”. (Los Angeles, California: ICANN, October 2015) https://www.icann.org/en/system/files/files/lgr-guidelines-second-level-27may20-en.pdf

[IANA] IANA Repository of IDN Practices, https://www.iana.org/domains/idn-tables

[IIS] IIS, IDN Reference Tables, https://github.com/dotse/IDN-tables, accessed on 10 October 2016

[Common] ICANN, Merged Second Level Reference Label Generation Rules, 30 November 2023 (XML) https://www.icann.org/sites/default/files/packages/lgr/lgr-second-level-common-24jan24-en.xml
non-normative HTML presentation:
https://www.icann.org/sites/default/files/packages/lgr/lgr-second-level-common-24jan24-en.html

[MSR] Integration Panel, "Maximal Starting Repertoire - MSR-5: Overview and Rationale", 24-June 2021, https://www.icann.org/en/system/files/files/msr-5-overview-24jun21-en.pdf

Note: MSR-5 is based on Unicode 11.0, the latest version for which IDNA 2008 tables are available at the time of its publication. Due to the fact that most recent additions to Unicode have been for rarely used code points, the probability that future versions will impact these reference LGRs is very low.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010, https://tools.ietf.org/html/rfc5890.

[RFC5891] J. Klensin, “Internationalized Domain Names in Applications (IDNA): Protocol”, RFC 5891, August 2010, https://www.rfc-editor.org/info/rfc5891

[RFC5982] Fältström, P., Ed., “The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)”, https://tools.ietf.org/html/rfc5892

[RFC6912] Sullivan, A., Thaler, D., Klensin, J., and O. Kolkman, "Principles for Unicode Code Point Inclusion in Labels in the DNS", RFC 6912, April 2013, https://tools.ietf.org/html/rfc6912.

[RFC7940] Davies, K and Asmus Freytag: “Representing Label Generation Rulesets using XML”, August 2016 https://tools.ietf.org/html/rfc7940.

[RFC8228] Asmus Freytag: "Guidance on Designing Label Generation Rulesets (LGRs) Supporting Variant Labels", RFC 8228, August 2017, https://www.rfc-editor.org/info/rfc8228.

[RZ-LGR-Overview] Integration Panel, "Root Zone Label Generation Rules - LGR-5: Overview and Summary", 26 May 2022 (PDF), https://www.icann.org/sites/default/files/lgr/rz-lgr-5-overview-26may22-en.pdf

[Second-Level Reference] ICANN, Second-Level Reference Label Generation Rules, https://www.icann.org/resources/pages/second-level-lgr-2015-06-21-en