# Integration Panel
# Packaging the MSR and RZ-LGR

VERSION 2017-09-15

*This document describes the Integration Panel's plans for packaging of the final integrated Root Zone LGR into a correlated set of XML files and some details about the XML file for the Maximal Starting Repertoire (MSR). To understand this document requires thorough familiarity with both the [Procedure] and the data format specification in RFC 7940. The Integration Panel is interested in getting feedback on this document from knowledgeable implementers and other consumers of its data tables.*

## 1 LGR

As described in the [Procedure], the Label Generation Ruleset for the Root Zone (RZ-LGR) will not simply be a single LGR but instead consist of a collection of integrated, script-specific LGRs having separate, if possibly overlapping repertoires, together with a common definition of non-reflexive variant relations.The assigned dispositions for variant labels are script-specific, but a in the common file all variants result in blocked labels, as appropriate for collision testing. In addition, the common file will contain a common set of Whole Label Evaluation (WLE) rules. This design has consequences for how the Root Zone LGR is best packaged into XML files according to [RFC 7940].

### 1.1 How the LGR will be packaged and details on file contents

The LGR will be packaged in a set of N+1 files, consisting of N script-specific files and a single merged Root Zone LGR file that is a composite of the N files.[1]

#### 1.1.1 Script-specific files

The formal specification of the submission requirements for a script-specific LGR, including all ancillary documentation is found in [SubmissionRequirement]. From these, the Integration Panel will derive both the merged file and the individual, script-specific files of which the Root Zone LGR is composed. (See Section 6 for a detailed specification for these script-specific files).

#### 1.1.2 Merged Root Zone File

The Integration Panel will prepare one merged file. Its format will match that of the script-specific files, except as follows. The file will contain:

1. Multiple <language> elements, one language element for each script represented
2. An integrated superset of all repertoire elements (without any “ref” attributes))
3. A union of the “tag” attribute values, if referenced by any WLE rules, may be present
4. An integrated superset of all variant mappings, except for reflexive variants

[1] This merged file may be referred to as “common” in file and other naming conventions.

5. No “type” attributes on variants
6. An integrated set of all WLE rules (including the Default Whole Label Evaluation rules from the MSR and corresponding actions triggered by them)

By definition, this merged file cannot describe all aspects of the Root Zone LGR. However, it can be used for several basic validation tasks, for example to verify that the total repertoire is a subset of the MSR (see below) or that all script files are compatible in their definition of variants and use of WLE rules. A full validity check for a given label, as well as determining the allocatable variants however, does require using the appropriate script-specific file. These operations require knowing the script for which the label is intended, because repertoires are defined per-script, as are the rules for assigning the disposition of variant labels. After a label is validated, the merged LGR is used to check for collisions to already delegated labels and their variants.

# 2 MSR

Under the [Procedure], the Maximal Starting Repertoire (MSR) is intended as input to the Generation Panels. It is accompanied by the Default Whole Label Evaluation (WLE) rules which, for practical purposes, are packaged in the same XML file as the definition of maximal repertoire. This section describes the contents of the data file for the MSR and key conventions used to represent it in XML.

## 2.1 Repertoire

The MSR contains neither variants nor variant dispositions, so <char> elements do not have child-elements. Each code point entry (<char> element) is tagged with a script tag. The tag is of the form “sc:xxxx” where “xxxx” is one of the elements of the Unicode Script_Extensions property value for the code point. Some code points that may be used with multiple scripts (such as combining marks) may be tagged with multiple script tags separated by space or simply with a tag for the pseudo script value Inherited: "sc:Zinh". For purposes of tagging, collective script IDs, such as “Jpan” are not used. Instead, any code point that belongs to such a collective script will be tagged with the applicable element scripts. For “Jpan” these would be sc:Hira, sc:Kana or sc:Hani, for example.

Consecutive <char> elements with the same tag values may be combined into <range> elements.

Each <char> (or <range>) element in the MSR is associated with a "ref" attribute giving the Unicode version for which the code point was first assigned a character. This information is provided for the convenience of the Generation Panels. This also serves to identify, which code points were encoded more recently, and therefore potentially not be as stable as those for which the encoding is of a more long-standing nature.

A <language> element will be set for each script supported by the MSR, but not for the pseudo script value of “Inherited”.

This resulting MSR data file is a single XML file; each Generation panel's maximal starting repertoire is the subset of those code points tagged with its scripts and relevant code points tagged as “Common” (Zyyy) or “Inherited” (Zinh).

## 2.2 Default Whole Label Evaluation Rules

The <rules> element in the MSR contains default WLE rules and default actions. These are not intended to allow the MSR itself to be used to validate labels, or determine dispositions — instead, they are to be added to each Generation Panels LGR. The GP may precede or intersperse them with proposed additional rules and actions, but not modify or delete them.

# 3 Constraints on Packaging and Options Investigated

The approach of packaging the result of the Root Zone LGR integration into separate files, as described above, is the result of an investigation by the Integration Panel on how to best satisfy the following constraints.

## 3.1 Constraints

The primary constraint is that, while the definition of variant (and the mappings from source code point sequence to target variant code point sequence) are universal for the Root, neither the allowed (sub-) repertoire for a label nor the dispositions for its variant labels are shared — they are specific to each of the script LGRs.

At the same time, the XML format for Representing Label Generation Rulesets allows only a single disposition per variant label, and has no way to tag individual elements by script. However, an entire file can be tagged with a script.

This makes packaging the LGR as a collection of per-script files the only practical option. It does present a complication in cases where a script-specific file defines any variant mapping that happens to cross script-specific sub-repertoire boundaries; the complication arises from the requirement that all variant relations be symmetric and transitive. The Integration panel investigated various implementations that would satisfy these additional constraints.

## 3.2 N+1 Files

During the processing of an application, the applied for label needs to be validated against the sub-repertoire associated with the script tag selected for the application. The only way to realize this with the XML format is to create a set of files, one per script tag, and then select the appropriate file to match the script tag of the applied for label. Likewise, the determination of allocatable variants depends on the per-script type values on the variant mappings combined with <action> elements. As for the repertoire, the <language> element at the head of the file specifies for which script tag the variant dispositions apply.

As a result, only a single one of the script-specific files needs to be accessed for the processing of any application, and the application's script tag will define which one of the files to use in processing the label for validity or in determining any allocatable variant labels.

The merged file of repertoire and variant mappings is to be used for collision checking and to verify that all script-specific files agree in their definitions of variant mappings.

In checking collisions between labels (and all their variants) each application would have to be evaluated first against the merged file to check for collisions, and then against the script-specific file to determine any allocatable variants.

## 3.3 How to represent variants that cross repertoire boundaries?

The various script-specific repertoires that make up the LGR are not strictly disjoint. That means that there is the possibility of variant mappings that cross repertoire boundaries. In addition, blocked variant mappings may exist between related scripts.

The Integration Panel investigated several possible options:

1. The <char> elements that are outside the repertoire are left out of the file, but any out-of-repertoire variant mappings that target these <char> elements are retained. The advantage is that the repertoire would trivially match the list of <char> elements, but then the files would no longer be symmetric and transitive. Retaining the mappings allows the files to be made symmetric and transitive mechanically, but performing that operation would result adding <char> elements in the XML that are again outside the repertoire.
2. Like 1 but without retaining variant mappings that map to a target code point outside the repertoire in the script-specific file, based on the notion that they are not needed for conflict checking because the merged file is used for that purpose. The script-specific files would no longer be symmetric and transitive, and cannot be used for to determine blocked variants outside the repertoire.
3. All <char> and <var> elements are retained in the script-specific file, such that the file is fully transitive and symmetric in its variant mappings. However, the out-of-repertoire <char> elements are identified as such, by giving them a reflexive variant mapping (identity mapping) with a type of "out-of-repertoire-var". Any variant mapping from any in-repertoire code point to an out-of-repertoire code point is assigned the type "blocked". Both of these operations can be done mechanically at the time the table is made transitive and symmetric. Finally, an action is defined in the XML file that resolves as "invalid" any label containing such out-of-repertoire code points, sequences or variants. That action looks like this:

```
<action disp="invalid" any-variant="out-of-repertoire-var" />
```

The final option (3) is the one preferred, and will be used in the Root Zone LGR. Using it, all the tables can be made formally symmetric and transitive — an essential point if there ever is a case where two code points are variants only because of transitivity involving an out-of-repertoire code point as the intermediate, as in

$$A \rightarrow O \text{ and } O \rightarrow B \Rightarrow A \rightarrow B$$

with code point O out of repertoire and A and B within the repertoire. If O and mappings to O were to be stripped from the script table, then that table could no longer be used to generate all variants and the variant dispositions can no longer be evaluated correctly.

The use of the reflexive variant as the means of identifying out-of-repertoire <char> elements allows a single, script-independent action to filter them.[2] That <action> element is part of the Default WLE specified in the MSR and thus is added by default to each script-specific file, whether or not the file contains any out of repertoire mappings; in other words, it has been made part of the default actions for the LGR. By creating the file with all entries needed to make it transitive and symmetric ***within*** the script-specific repertoire, but still containing all variant mappings to out-of-repertoire <char> elements, a tool can be used to add the needed <char> elements and variant mappings and set the correct dispositions mechanically.

In essence, option 1 (or in some cases 2) describes how a Generation Panel would structure the initial drafts of its LGR proposal. Using option 1 or 2 describes most clearly the linguistic intent from the point of view of the given script, but such drafts are incomplete from the point of view of the Root Zone LGR and cannot be used to determine the full set of variant labels needed for collision testing. Option 3 describes what the corresponding script-specific file will look like as result of LGR integration and is the only complete specification from which to derive the set of variant labels. Generation Panels are strongly encouraged to create such a full specification and submit it to public comment as part of their LGR proposal to allow reviewers to anticipate how labels will actually be processed. (In the absence of out-of-repertoire variant mappings, of course, all of these options are the same).

### 3.4 Variants in the Merged XML file

The merged file will contain a combined set of all <char> and <var> elements, with the proviso that the "type" attribute will be set to "blocked". That way, the merged file can be used for collision checking.

In the merged file, no <char> element is out of repertoire. Therefore, all reflexive "out-of-repertoire-var" mappings are removed as part of the merged set.

Any other reflexive mappings are also removed from the merged set. The purpose of a reflexive mapping consists entirely of allowing a "type" value to be associated with an original code point, so that this type value can be used in determining label validity. Like the special case of "out-of-repertoire-var", these "type" values are specific to the script-LGR, but unlike regular variants there is no suitable default value that can be assigned; that means leaving reflexive <var> elements out of the merged set.

## 4 Permissible reformulation of script LGRs' XML files

The integrated LGR will not consist of a collection of verbatim copies of the XML files for the LGR proposals as submitted by the Generation Panels. Instead, these XML files may be reformatted and reorganized for consistency or re-expressed with equivalent statements, provided that the reformulated version is equivalent. Equivalence of two formulations of an LGR is solely defined by whether they result in the same namespace of valid labels and same namespace of allocatable or blocked variant labels.

[2] The Integration Panel explored various alternatives for identifying out-of-repertoire code points, but in the context of the [RFC 7940] the use of a special type of reflexive variant, while perhaps initially not the most intuitive, proved ultimately the most robust and simple.

## 4.1 Informative elements and attributes

Informative elements and attributes (comments, references, and descriptions) contained in the submitted XML files may or may not be preserved in the integrated LGR. These include tag values not referenced in any <class> elements. Comments on the XML source level will be discarded. Whether retained in the final LGR or not, all informative data remain a matter of record as part of the original submission. Additional informative items may be added as result of integration.

The precise value of name attributes for <rule> and <class> elements are informative and may be adjusted. Likewise, tag values may be renamed and references renumbered.

## 4.2 Non-default variant types

These may be renamed.

## 4.3 Whole Label Evaluation Rules

Whole label evaluation rules are not script-specific. The Integration panel will create a merged statement of all <rule> and <action> elements for the merged file, using the provided test cases to ensure that the reformulation produces the same dispositions for variant labels.

### 4.3.1 The <rule> and <class> elements

Default rule and class names in the merged file will be prefixed with “Common-”. Other rule and class names will be prefixed with a script identifier, e.g. “Arab-“. This will prevent name collisions in the merged set of rules and classes. Additional manual adjustments of rule and class names may be made.

### 4.3.2 The <action> elements

A merged set of <action> elements will be created, including the Defaults Actions defined in the MSR. The relative order of <action> elements defines their precedence. The order of precedence may be adjusted manually as result of testing.

# 5 Script-specific files

The following provides a detailed specification for the script-specific XML files that are part of the Root Zone LGR. For the actual syntax level specification see [RFC7940].

## 5.1 The <meta> element

- Each file contains a meta element with these elements
  - <version>
  - <date>
  - <language>
  - <scope>
  - <unicode-version>
  - <description>

- The <version> element is set to the version of the Root Zone LGR
- The <date> element is set to the release date of the Root Zone LGR

- The single <language> element is in the format “und-{script}” where {script} is the ISO 15924 alphanumeric script code for the file, e.g. “und-Cyrl” or “und-Jpan”.
- The single scope element has a “type” attribute of “domain” and is set to “.” to indicate the root zone.
- The Unicode version element is set to the Unicode version number the LGR is based on, e.g. “6.3.0”. This matches the Unicode version number of the MSR version that the LGR is based on.
- The <description> element gives some summary information, in particular information relevant to understanding the file itself. If the file uses non-default values for the “type” attributes on variants, these are summarized in the description. A brief description of any <action> or <rule> element defined in the file (for WLE rules) are also provided.
- A <references> element giving the references used. [NOTE: in the MSR the IP documented the Unicode version for each code point; this information is useful in analyzing the repertoire. It may also be useful for the LGR, and as such is retained.]

## 5.2 The <data> element

The <data> element lists the repertoire for the script and any variants defined for it.

### 5.2.1 Repertoire

The repertoire is listed by <char> or <range> elements.

- Each file will contain a <char> element for each member of its repertoire, except that any adjacent <char> elements may be collapsed into <range> elements.
- Additional, out-of-repertoire <char> elements are added when needed for a fully symmetric and transitive table because variants are defined that map outside the repertoire. Any such out-of-repertoire <char> elements will have a <var> element defining a reflexive variant with type="out-of-repertoire-var" to identify the code point as not part of the repertoire.
- All <char> and <range> elements will be tagged with script values following the scheme in the MSR.

### 5.2.2 Variants

Variant mappings in each file are symmetric and transitive.

- Variants are specified with <var> elements. Any <var> element with a target cp that is out of repertoire will have type="blocked", except where the type is set to “out-of-repertoire-var” for a reflexive mapping.
- Default type attributes of other variants are "blocked" or "allocatable"; unless non-default values are used, in which case the <rules> element will contain <action> elements that evaluate such values into either a "blocked" or an "allocatable" disposition for any label containing that variant.
- Reflexive variants may be used as part of a scheme to limit the number of allocatable variants, or, as described, to identify out-of-repertoire code points.

## 5.3 Whole Label Evaluation

The Whole Label Evaluation (WLE) rules are common (that is, not script-specific) for the LGR. Where repertoires overlap, all whole label evaluation rules and actions triggered by them that could affect the same label must be consolidated, so they appear in the same order of precedence, with corresponding <rule> and <class> elements named the same; the values of relevant “tag” and “type” attributes must also match. Where repertoires are disjoint, rules that can only be triggered for a specific repertoire may not need to be adjusted

### 5.3.1 The <rule> and <class> elements

Script-specific LGRs will only list those rules actually affecting labels in that script, as well as all Default Rules from the MSR.

### 5.3.2 The <action> elements

Script-specific LGRs will only list those actions actually affecting labels in that script, as well as all Default Actions from the MSR.

# 6 Other Files

For each XML file, a mechanically generated HTML file will be provided that gives a more human-readable presentation of the LGR. This file may include additional informative data, such as counts of elements, Unicode Character Names and similar. (The XML file remains the normative specification of the LGR).

One of more PDF files showing the repertoire in a format similar to the Unicode Code charts may be provided.

Finally, the LGR and MSR will each have a single overview document that lists all the constituent files and presents a summary and discussion of the main features. For the RZ-LGR, this file will also give a list of the constituent proposals from which the RZ-LGR was derived.

# 7 References

[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." (Los Angeles, California: ICANN, March, 2013) http://www.icann.org/en/resources/idn/variant-tlds/draft-lgr-procedure-20mar13-en.pdf

[RFC7940] Davies, K. and A. Freytag, "Representing Label Generation Rulesets using XML", RFC 7940 https://tools.ietf.org/html/rfc7940.

[RFC8228] Freytag, A., “Guidance on Designing Label Generation Rulesets (LGRs) Supporting Variants”, RFC 8228, https://www.rfc-editor.org/rfc/rfc8228.txt

[SubmissionRequirements] Integration Panel, “Requirements for LGR Proposals”
https://community.icann.org/download/attachments/43989034/Requirements%20for%20LGR%20Proposals-2017-09-15.pdf

[WLE Rules] Integration Panel, “Whole Label Evaluation (WLE) Rules”
https://community.icann.org/download/attachments/43989034/Whole%20Label%20Evaluation%20Rules-2017-09-15.pdf