L2/19-364R

Feedback on Proposed Update UTS #18 2019-09-12

Markus Scherer 2019-oct-08

Properties of strings: Matching behavior

When strings are involved, there are different possible & reasonable behaviors for repeated matching as in \{PropOfStrings}+. In a regex, the most consistent would be to match any path, with backtracking or equivalent; that is, behaving as if it was a repeated OR expression – however that expression is matched in that regex engine. So a property that matches strings “ab”, “abc”, “cd”, together with the plus operator, would match the complete text “abcd”, exactly like a regex of “(abc|ab|cd)+” would. Note that, in order to account for strings with common prefixes, it should behave as if longer strings were provided first in the alternation.

Thus, the document should specify that semantically the property acts as an alternation sorted with longer strings first.

Problematic properties

Regarding **2.7 Full Properties** and its review note “The properties in SpecialCasing.txt such as Lowercase_Mapping are actually conditional, and should probably be removed since they require another argument (the condition).”:

Some of the Full Properties are not well-defined for regex matching. For example, the definition of [full] Lowercase_Mapping involves mappings conditional on language and/or context as specified in UCD SpecialCasing.txt. It is thus not a pure code point → string property. Language and context would form part of either the input or the output, and expected behavior in a regular expression is not obvious.

Please remove following properties:

> Lowercase_Mapping, Titlecase_Mapping, Uppercase_Mapping, Case_Folding

Please also remove related examples from 2.8 Optional Properties, such as

[:toLowercase=a:]	The set of all strings X such that toLowercase(X) = "a"

Unclear suggested properties

Regarding **2.8 Optional Properties** “Implementations may also add other regular expression properties based on Unicode data that are not listed above. Some possible candidates include the following.”:

- Exemplar characters from [UTS35]
- IDNA status and mapping from [UTS46]

→ Syntax for these “possible candidate properties” is only vaguely suggested. Implementers may come up with a wide range of how to express them. I propose to add more examples/suggestions, such as:

- Exemplar characters for particular locales from [UTS35], such as \p{Exemplar=fr:main}
  - *Note: Two dimensions of parameters: locale & main|aux|punctuation|index|...*
- IDNA status and mapping from [UTS46], such as \p{UTS46_Status=deviation}

Identifier_Status and Identifier_Type from [UTS39]

→ See the next section of this document about naming issues.

Properties defined outside the UCD

Regarding **1.2 Properties**: For use in regular expressions, properties can also be considered to be defined by Unicode definitions and algorithms, ... Other Unicode Technical Standards, such as *UTS #51 Unicode Emoji*, provide names for definitions and algorithms that can be used for the names of regular expression properties.

In my opinion, this is useful but underspecified. For UCD properties, we have UCD data files that provide the names and aliases of both the properties and their values (if enumerated/catalog).

An implementation of UTS #18 or similar (including ICU UnicodeSet) uses a unified namespace of properties. For non-UCD properties, this means that there must not be collisions of property names and aliases with UCD property names and aliases (or values, where the property name is often omitted, such as for Script and General_Category).

The problem is that non-UCD properties tend not to have formally documented names and aliases of properties and values. Even the property *types* may not be obvious (e.g., set of binary properties vs. one property with enumerated values).

I suggest that we add one or more data files that document the names and aliases (and probably types) of non-UCD properties. The location could be among the UCD files, or separately, e.g., associated with UTS #18.

There are two options:

Combined property names file

We could add a single file with simple syntax like in ICU ppucd.txt that provides names and aliases of both properties and their values, together with and explicit field for the property type, as follows:

OtherProperties.txt

```
# UTS #39 properties
property;Enumerated;IdentifierStatus;IdentifierStatus  # short & long names are the same
property;Enumerated;IdentifierType;IdentifierType

value;IdentifierStatus;Inclusion;Inclusion
value;IdentifierStatus;Recommended;Recommended
value;IdentifierType;Not_Character;Not_Character
value;IdentifierType;Deprecated;Deprecated
value;IdentifierType;Default_Ignorable;Default_Ignorable
value;IdentifierType;Not_NFKC;Not_NFKC
value;IdentifierType;Not_XID;Not_XID
value;IdentifierType;Exclusion;Exclusion
value;IdentifierType;Obsolete;Obsolete
value;IdentifierType;Technical;Technical
value;IdentifierType;Uncommon_Use;Uncommon_Use
value;IdentifierType;Limited_Use;Limited_Use
```

Separate properties vs. values names files

Alternatively, we could add a new pair of files with the syntax of UCD Property*Aliases.txt. On the positive side, existing parsers could be easily adapted to just read multiple sets of files. On the negative side, separate files are clunkier, and the property type is only in a comment (a defect shared by the current PropertyAliases.txt file).

OtherPropertyAliases.txt

```
# ================================================
# Enumerated Properties
# ================================================
IdentifierStatus ; IdentifierStatus
IdentifierType ; IdentifierType
```

OtherPropertyValueAliases.txt

```
# IdentifierStatus

IdentifierStatus ; Inclusion ; Inclusion
IdentifierStatus ; Recommended ; Recommended

# IdentifierType
```

```
IdentifierType ; Not_Character ; Not_Character
IdentifierType ; Deprecated ; Deprecated
IdentifierType ; Default_Ignorable ; Default_Ignorable
IdentifierType ; Not_NFKC ; Not_NFKC
IdentifierType ; Not_XID ; Not_XID
IdentifierType ; Exclusion ; Exclusion
IdentifierType ; Obsolete ; Obsolete
IdentifierType ; Technical ; Technical
IdentifierType ; Uncommon_Use ; Uncommon_Use
IdentifierType ; Limited_Use ; Limited_Use
```

Editorial issues

Examples of such syntax are \p{Script=Greek} and [:Script=Greek:], which stands for the set of characters that have the Script value of Greek.

→ Make “stands” plural. Maybe even “which both stand for”.

A property value can also be a *set* of values. For example, the Script_Extensions property maps from code points to a set of enumerated Script values, ...

→ In the following text, there are several instances of singular Script_Extension rather than plural Script_Extensions.

Other Unicode Technical Standards, such as *UTS #51 Unicode Emoji*, provide names for definitions and algorithms that can be used for the names of regular expression properties.

→ (Most of the?) emoji properties are now in the UCD.

Surrogate pairs (or their equivalents in other encoding forms) are be handled internally as single code point values.

→ Fix grammar around “are be handled”.

Add underscore to UTS #39 Identifier properties

The example above uses the property name spelling in UTS #39, without underscore where UCD properties customarily include an underscore (which would look like “Identifier_Status”).

Even though the underscore is ignored in property name matching, it is cleaner to include it in the formal name, for consistency. Therefore, I propose changing the spelling of the UTS #39 properties to Identifier_Status and Identifier_Type.