L2/20-109

UTC #163 regex feedback & recommendations

Markus Scherer et al., 2020-apr-21

PRI: www.unicode.org/review/pri404/ “Proposed Update UTS #18, Unicode Regular Expressions”
Latest draft: www.unicode.org/reports/tr18/tr18-20.html

M0: Recommended UTC actions

1. AI for Rick: Close PRI #404.
2. Motion: Approve UTS #18, with changes recommended in L2/20-109, for publication as version 21.
3. AI for Mark, Markus and the ed committee: For UTS #18: Apply the changes recommended in L2/20-109.

Feedback reported to Unicode

None as of 2020-apr-20

Other feedback/discussion

F1: Removal of C3

Markus: In 0.2 Conformance, clause C3 for level 3 is being removed, and C4 is renumbered to C3.
I think this is too confusing -- I think we should leave an empty stub for C3 and keep C4 numbered as is.

Recommended UTC action

1. AI for Mark: In UTS #18 section 0.2 Conformance, leave an empty stub for C3 and keep C4 numbered as is.

F2: Negation of set difference

Andy: A question about the examples of resolving character classes with strings, from Annex D

[^C--S] is not valid

I don't see why not. [C--S] can contain no strings, so negating it with [^C--S] would be safe.

Recommended UTC action

1. AI for Mark: In UTS #18 Annex D, replace the fourth bulleted character class example as recommended in L2/20-109.

Recommended text change

Proposed: Change the fourth bulleted character class example to

- [^C--S] is allowable
- [^S--C] is not valid

FYI

Andy actually referred to “Annex E”, but the proposed Annex D has been removed since then, and Annex E has been renumbered.

E0: Small editorial issues

Recommended UTC action

1. AI for Mark, Markus, and the ed committee: In UTS #18: Incorporate editorial feedback from L2/20-109 (item E0 and below) as appropriate.

E1.3

Markus: In 1.3 Subtraction and Intersection: The symmetric difference is listed as one of the operators required for RL1.3, but the following paragraph makes it sounds like an additional option: “Implementations may also choose to offer other set operations. The symmetric difference of two sets is also useful. It is defined as ...”

Mark: The first line is outdated. I suggest resolving the text as follows:
OLD

…

:= "~~" // symmetric difference: = (A\B)∪(B\A) = (A∪B)\(A∩B)

Implementations may also choose to offer other set operations. The symmetric difference of two sets is also useful. It is defined as being the union minus the intersection. Thus **`[\p{letter}~~\p{ascii}]`** is equivalent to **`[[\p{letter}\p{ascii}]--[\p{letter}&&\p{ascii}]]`**.

NEW

…

:= "~~" // symmetric difference: = (A∪B)\(A∩B)

The symmetric difference of two sets is defined as being the union minus the intersection, that is (A∪B)\(A∩B), or equivalently, the union of the asymmetric differences (A\B)∪(B\A).

THEN move the example to the table of examples. that is, below

[\p{Greek}--\N{GREEK SMALL LETTER ALPHA}]Greek letters except alpha**[\p{Assigned}--\p{Decimal Digit Number}--a-fA-Fａ-ｆＡ-Ｆ]**all assigned characters except for hex digits (using a broad definition)

[\p{Assigned}--\p{Decimal Digit Number}--a-fA-Fａ-ｆＡ-Ｆ]	all assigned characters except for hex digits (using a broad definition)
[\p{letter}~~\p{ascii}]	the letters that are not ASCII and the ASCII that are not letters. So Letters and ASCII, minus [A-Za-z]

E2.5.1

Markus: In 2.5.1 Individually Named Characters: Remove “in” from “... which can thus be used in wherever \u{...} can be used.”

E2.8a

Markus: Fix typo in 2.8 Optional Properties, named sequence example description “... is a drop-ins for ...” (remove ‘s’)

E2.8b

Markus: In 2.8 Optional Properties the last two examples are missing values: \p{Indic_Positional_Category} & \p{Indic_Syllabic_Category} They don’t make sense as is. Please add values to the examples, such as \p{Indic_Positional_Category=Left_And_Right} & \p{Indic_Syllabic_Category=Avagraha}

EA

Markus: In Annex A: Character Blocks, table Writing Systems Versus Blocks: L2/20-051 had an action item to “Writing Systems Versus Blocks ⇒ add extension F & G” which is not done yet.

EDa

Markus: In Annex D: Resolving Character Classes with Strings: L2/20-051 had an action item to “Next Rev, add symmetric diff to table in Annex E” which is not done yet.

EDb

Mark: Fix typo in Annex D: Resolving Character Classes with Strings: [\p{Basic_Emoji}-\p{any}] should be [\p{Basic_Emoji}&&\p{any}]

EDc

Markus: In Annex D, the first part of the following sentence seems unclear (what is a “character class with characters”?), and the last part seems wrong (no negated class with strings). I : “The following describes how a boolean expression can be resolved to a Character Class with characters, a Character Class with strings, or a negated Character Class with strings.”

→ Suggestion: “The following describes how a boolean expression can be resolved to a Character Class with only characters, a Character Class with strings, or a negated Character Class with only characters.”

EEa

Markus: In Annex E: Notation for Properties of Strings: “As described in Annex E, some character class expressions are invalid when they contain properties of strings. Detection of such invalid expressions should be happen early, when the regular expression is first compiled or processed.”
→ Change “Annex E” to “Annex D”
(The earlier Annex D was for the properties metadata file; that annex has been removed and the following ones renumbered.)
→ Remove “be” from “should be happen early”

EEb

Michael/Markus: In Annex E: Notation for Properties of Strings: “`\m` should also accept ordinary properties of characters; it can be limited in where it may appear, not in what properties it allows.”
→ This should be a separate paragraph, reworded like this:

> \m should also accept ordinary properties of characters. If a property that applies to strings later changes to only apply to characters, a regex with such a \m{property} should not become invalid. Also, being able to use the same \m syntax outside of a character class for any property would be easy for a regex writer.

EMa

Markus: In Modifications: “Updated the full property list to include newer UCD properties plus Emoji properties and UTS #39 properties, and dded a data file with property metadata for supporting non-UCD properties. ...” → remove the last part starting with “, and dded a data file…”

EMb

Markus: Near the end of Modifications: Fix typo in “Removed examples of toNFIC_CaseFold, CJK, and other properties” (K not I in the first function name)

EMc

Markus: At the end of Modifications: Also add “Added Annex E: Notation for Properties of Strings”

E1.2 Rewrite of section 1.2 Properties

Ken: I now have a complete, suggested re-draft of Section 1.2 of the document […] No way to make a bunch of point comments for something like this. I've just rewritten the whole [...] thing into something that makes more sense, and with better presentation of the tables of examples.

Note that in this draft, all the text after the first two paragraphs is new text, so should all be in "changed" style -- I just haven't bothered for this particular copy, because it just makes it harder to read for review.

Replacement text for section 1.2 from Ken & Mark (& light edits from Markus):

1.2 Properties

Because Unicode is a large character set that is regularly extended, a regular expression engine needs to provide for the recognition of whole categories of characters as well as simply literal sets of characters and strings; otherwise the listing of characters becomes impractical, out of date, and error-prone. This is done by providing syntax for sets of characters based on the Unicode character properties, as well as related properties and functions. Examples of such syntax are \p{Script=Greek} and [:Script=Greek:], which both stand for the set of characters that have the Script value of Greek. In addition to the basic syntax, regex engines also need to allow them to be combined with other sets defined by properties or with literal sets of characters and strings. An example is [\p{Script=Greek}-\p{General_Category=Letter}], which stands for the set of characters that have the Script value of Greek *and* that do not have the General_Category value of Letter.

Many character properties are defined in the Unicode Character Database (UCD), which also provides the official data for mapping Unicode characters (and code points) to property values. See ~~Section 2.7,~~ *Full Properties*~~;~~ UAX #44, *Unicode Character Database* [UAX44]~~;~~ and Chapter 4 in *The Unicode Standard* [Unicode]. For use in regular expressions, properties can also be considered to be defined by Unicode definitions and algorithms, and by data files and definitions associated with other Unicode Technical Standards, such as UTS #51, *Unicode Emoji*. For example, this includes the Basic_Emoji definition from UTS #51. The full list of recommended properties is in Section 2.7 Full Properties. ~~the defined Unicode string functions, such as isNFC() and isLowercase(), which also apply to single code points and may be useful to support in regular expressions.~~

UAX #44, *Unicode Character Database* [UAX44] divides character properties into several types: Catalog, Enumeration, Binary, String, Numeric, and Miscellaneous. Those categories are not all precisely defined or immediately relevant to regular expressions. Some are more pertinent to the maintenance of the Unicode Character Database.

1.2.1 Domain of Properties

For regular expressions, it is more helpful to divide up properties by the treatment of their domain (what they are properties of) and their codomain (the values of the properties). Most properties are properties of Unicode code points; thus the domain is simply the full set of Unicode code points. Typically the important information is for the subset of the code points that are characters; therefore, those properties are often also called properties of characters.

In addition to properties of characters, there are also properties of strings (sequences of characters). A property of strings is more general than a property of characters. In other words, any property of characters is also a property of strings; its domain is, however, limited to strings consisting of a single character.

Data, definitions, and properties defined by the Unicode Standard and other Unicode Technical Standards, which map from strings to values, can thus be specified in this document as defining regular-expression properties.

Negations of properties of strings or Character Classes with strings may not be valid in regular expressions. For more information, see *Annex D: Resolving Character Classes with Strings* and *Section 2.2.1 Character Classes with Strings*.

1.2.2 Codomain (Values) of Properties

The values (codomain) of properties of characters (or strings) have the following simple types: Binary, Enumerated, Numeric, Code Point, and String. Properties can also have multivalued types: a Set or List of other types.

The Binary type is a special case of an Enumerated type limited to precisely the two values "True" and "False". In general, a property of Enumerated type has a longer list of defined values. Those defined values are abstractions, but they are identified in the Unicode Character Database with labels known as aliases. Thus, the Script value "Devanagari" may also be identified by the abbreviated alias "Deva"—both refer to the same enumerated value, even though the exact label for that value may differ.

The Code Point type is a special case of a String type where the values are always limited to single-code point strings.

The UCD "Catalog" type is the same as Enumerated (the name differs for historical reasons).

1.2.3 Examples of Properties

The following tables provide some examples of property values for each domain type.

Examples of Properties of Characters

Type	**Property Name**	**Code Point**	**Character**	**Value**	**Regex Literal**
Binary	White_Space	U+0020	" "	True	
	Emoji	U+231A	⌚	True	
Enumerated	Script	U+3032	〲	Common	
Code point	Simple_Lowercase_Mapping	U+0041	A	"a"	\u{61}
String	Name	U+0020	" "	"SPACE"	\u{53 50 41 43 45}
Set	Script_Extensions	U+3032	〲	{Hira, Kana}	

Note: The Script_Extensions property maps from code points to a set of enumerated Script property values.

Expressions involving Set properties, which have multiple values, are most often tested for containment, not equality. An expression like \p{Script_Extensions=Hira} is interpreted as containment: matching each code point *cp* such that Script_Extensions(*cp*) ⊇ {Hira}. Thus, \p{Script_Extensions=Hira} will match both U+3032 〲 VERTICAL KANA REPEAT WITH VOICED SOUND MARK (with value {Hira Kana}) and U+3041 ぁ HIRAGANA LETTER SMALL A (with value {Hira}). That also allows the natural replacement of the regular expression \p{Script=Hira} by \p{Script_Extensions=Hira}—the latter just adds characters that may be *either*

Hira *or* some other script. For a more detailed example, see *Section 1.2.2 Script and Script Extensions Properties*.

Expressions involving List properties may be tested for containment, but may have different semantics for the elements based on position. For example, each value of the kMandarin property is a list of up to two String values: the first being preferred for zh-Hans and the second for zh-Hant (where the preference differs).

Examples of Properties of Strings

Type	**Property Name**	**Code Point(s)**	**Character(s)**	**CLDR Name**	**Value**
Binary	Basic_Emoji	U+231A	⌚	watch	True
		U+23F2 U+FE0F	⏲️	timer clock	True
		U+0041	A		False
		U+0041 U+0042	"AB"		False
	RGI_Emoji_Flag_Sequence	U+1F1EB U+1F1F7	🇫🇷	flag: France	True

Note: Properties of strings can always be "narrowed" to just contain code points. For example, [\p{Basic_Emoji} && \p{any}] is the set of characters in Basic_Emoji.

EDT: Editorial: Clearer table for Annex D

Markus: I find the presentation in the Annex D table confusing, especially the "action" which retains A's flag if the flag is not mentioned.

Markus & Mark replacement:

[expression] and \p{expression} (without negation) create enhanced sets with the sets corresponding to the expression, and the flags set to true.

[^expression] and \P{expression} (with negation) create enhanced sets with the sets corresponding to the expression, and the flags set to false.

[^A] where A is an enhanced set with (set, flag) results in the flag being inverted.

Binary operations are resolved as follows. The symbol ➕ stands for the flag value true/normal. The symbol ➖ stands for the flag value false/negative.

Syntax	**Flag of A**	**Flag of B**	**Result Set**	**Flag of Result**
A || B	+	+	setA ∪ setB	+
	+	−	setB \ setA	−
	−	+	setA \ setB	−
	−	−	setA ∩ setB	−
A && B	+	+	setA ∩ setB	+
	+	−	setA \ setB	+
	−	+	setB \ setA	+
	−	−	setA ∪ setB	−
A -- B	+	+	setA \ setB	+
	+	−	setA ∩ setB	+
	−	+	setA ∪ setB	−
	−	−	setB \ setA	+

Note: the operation A ~~ B can be handled in terms of other operations according to its definition: (A∪B)\(A∩B)