L2/21-003

UTS #18 Editorial additions: EBNF

M. Davis, 2021-01-XX

This is a continuation of L2/21-002 UTS #18 Editorial additions, a proposal for changes to https://unicode.org/reports/tr18/. It is broken out because a landscape view makes it much easier to see these changes. This is simply for review; the final text in UTS #18 would not be this wide: the 2nd two columns will fit in a normal page width.

Problem: There is EBNF used to describe the structure of a Character Class. It needs some cleanup, because there are some omissions, typos, and stylistic inconsistencies. (These were obscured because the EBNF is split by section, as each feature is introduced.)

Proposal: the old text (OLD) would be replaced by a table with the second & third columns, in each respective section, applying any necessary cleanup to references to the EBNF terms. Note that the precise syntax is not normative in UTS #18, since it is typically adapted to the current syntax of any particular Regex Engine. The new EBNF may deviate from the particulars of the old EBNF.

OLD	NEW	Comments
https://unicode.org/reports/tr18/#character_ranges		
 CHARACTER_CLASS := "[" NEGATION? ITEM (OPERATOR? ITEM)* "]" ITEM := "[" CHARACTER_CLASS "]" := CODE_POINT2 := CODE_POINT2 "-" CODE_POINT2 // range CODE_POINT2 := ESCAPE CODE_POINT := CODE_POINT NEGATION := "^" OPERATOR := "" // union (no separator): A∪B := "||" // union (...): A∪B ESCAPE := "\"	CHARACTER_CLASS := '[' NEGATION? SEQUENCE ']' SEQUENCE := ITEM+ ITEM := LITERAL ('-' LITERAL)? := CHARACTER_CLASS LITERAL := ESCAPE (SYNTAX_CHAR | SPECIAL_CHAR) := NON_SYNTAX_CHAR := HEX NEGATION := '^' ESCAPE := '\' SYNTAX_CHAR := [\-\[\]\}\/\\\^] SPECIAL_CHAR := [abcefnrtu] NON_SYNTAX_CHAR := [^SYNTAX_CHAR] SP := ' '+	• NEGATION is also known as *complement* • union of items: A∪B... This is replaced with operators in RL1.3 Subtraction and Intersection • Constraint: parse error if in range with 1st literal ≥ 2nd literal (some Regex Engines may allow ==) • Different variants of SYNTAX_CHAR, SPECIAL_CHAR, and NON_SYNTAX_CHAR can be adjusted for particular contexts & compatibility • The exact set of SPECIAL_CHAR may vary across Regex engines. • [^SYNTAX_CHAR] means all valid Unicode code points except for those in SYNTAX_CHAR *REVIEW NOTE: The new rules severely limit the set of characters allowed to be ESCAPEd, compared with the old rules, more consistent with general practice.*
	Add below the rules: The EBNF can be modified for compatibility with existing syntax, or enhanced with other features. For example, to allow ignored spaces for readability, add \u{20} to SYNTAX_CHAR, and add SP? around various elements, change ITEM+ to SP? ITEM (SP? ITEM)+, etc. In subsequent sections of this document, additional EBNF lines will be added for additional features. In one case, marked in a comment, one of the above lines will be replaced.	

https://unicode.org/reports/tr18/#Hex_notation

```
<codepoint>   := <character>
<codepoint>   := "\u" HEX_CHAR HEX_CHAR HEX_CHAR HEX_CHAR
<codepoint>   := "\u{" HEX_CHAR+ "}"
<codepoints>  := "\u{" HEX_CHAR+ (SEP HEX_CHAR+)* "}"
SEP           := \s+
```

```
HEX             := '\u' HEX_CHAR{4}
                := '\u{' CODEPOINT (SP CODEPOINT)* '}'
HEX_CHAR        := [0-9A-Fa-f]
CODEPOINT       := '10' HEX_CHAR{4} | HEX_CHAR{1,5}
```

REVIEW NOTE: NEW is more precise about hex formats, & merges SEP into SP.

Add below: Note: \u{3b1 3b3 3b5 3b9} is just semantic sugar for \u{3b1}\u{3b3}\u{3b5}\u{3b9} — useful for readability and concision but not a requirement. Thus [a-\u{3b1 3b3}-ξ] behaves like [a-\u{3b1}\u{3b3}-ξ] == [a-αγ-ζ]

https://unicode.org/reports/tr18/#property_syntax

```
CHARACTER_CLASS := POSITIVE_SPEC | NEGATIVE_SPEC
ITEM            := POSITIVE_SPEC | NEGATIVE_SPEC
POSITIVE_SPEC   := ("\p{" PROP_SPEC "}") | ("[:" PROP_SPEC ":]")
NEGATIVE_SPEC   := ("\P{" PROP_SPEC "}") | ("[:^" PROP_SPEC ":]")
PROP_SPEC       := <binary_unicode_property>
PROP_SPEC       := <unicode_property> (":" | "=" | "≠" | "!=" )
PROP_VALUE
PROP_SPEC       := <script_or_category_property_value> ("|"
<script_or_category_property_value>)*
PROP_VALUE      := <unicode_property_value> ("|"
<unicode_property_value>)*
```

```
CHARACTER_CLASS  := '\' [pP] '{' PROP_SPEC '}'
                 := '[:' PROP_SPEC ':]'
PROP_SPEC        := PROP_NAME (RELATION PROP_VALUE)?
PROP_NAME        := ID_CHAR+
ID_CHAR          := [A-Za-z0-9\ \-_]
RELATION         := ':' | '=' | '≠' | '!='
PROP_VALUE       := LITERAL*
```

- *Adds to* previous CHARACTER_CLASS rules.

- Constraint: PROP_NAME = valid Unicode property name or alias, or extended property name or alias. See RL1.2 Properties, 2.7 Full Properties, RL2.7 Full Properties, and 2.8 Optional Properties

- Constraint: PROP_VALUE = valid Unicode property value for that PROP_NAME

https://unicode.org/reports/tr18/#Subtraction_and_Intersection

```
OPERATOR := "&&" // intersection: A∩B
         := "--" // set difference: A\B
         := "~~" // symmetric difference: (A∪B)\(A∩B)
```

```
SEQUENCE        := ITEM (SEQ_EXTEND)*
SEQ_EXTEND      := OPERATOR CHARACTER_CLASS | ITEM
OPERATOR        := '||'
                := '&&'
                := '--'
                := '~~'
```

- *Replaces* SEQUENCE definition above which has just ITEM+

- union: A∪B (explicit operator where desired for clarity)
- intersection: A∩B
- set difference: A\B
- symmetric difference: A⊖B = (A∪B)\(A∩B)

Add below the rules:

For better backwards compatibility (and clarity), this syntax requires a character class after an operator. So [ab&&[bc]] is valid, but [ab&&bc] is not. If there might be an ITEM to the right of the operator, [ab--cd] could be interpreted as [[ab-c]d]. So it reduces the ambiguity to require a CHARACTER_CLASS and not just an ITEM.

However, the exact way that operator precedence is handled may differ by regex implementation.

https://unicode.org/reports/tr18/#Character_Ranges_with_Strings

```
ITEM := "\q{" (CODE_POINT (SP CODE_POINT)*)? "}"
SP   := \u{20}
```

```
ITEM            := '\q{' LITERAL* '}'
```

- *Adds to* previous ITEM rules.
- literal string of characters

REVIEW NOTE: the OLD SP was a duplicate of SEP and is now merged

https://unicode.org/reports/tr18/#Individually_Named_Characters

```
<codepoint> := "\N{" <character_name> "}"
```

```
LITERAL         := '\N{' ID_CHAR+ '}'
```

- *Adds to* previous LITERAL rules.
- Constraint: ID_CHAR+ = valid Unicode name or alias

https://unicode.org/reports/tr18/#Wildcard_Properties

```
PROP_VALUE := <value>
            | "/" <regex expression> "/"
            | "@" <unicode_property> "@"
```

```
PROP_VALUE      := '/' <regex expression> '/'
                := '@' FUNCTION '@'
```

• Constraint: FUNCTION is a PROP_NAME or an extended function like toNFC.

Add in text below: For example, \p{name=/(SMILING|GRINNING)/} is the set of all characters whose name matches the expression, such as 🤩 U+1F929 GRINNING FACE WITH STAR EYES