L2/18-337

Broaden the scope of what Unicode calls “properties”

by Mathias Bynens (mths@google.com)

Proposal:

1. Officially broaden the scope of “properties” to include properties of strings as well as properties of code points.
2. Characterize 6 sets of emoji data currently called type_fields as properties of strings.
3. Adopt a policy that any new Unicode properties of strings have the `_Sequence` suffix, and rename the proposed `Basic_Emoji` property as `Basic_Emoji_Sequence` accordingly.

Problem:

In Unicode terms, the word “properties” is currently restricted to just properties of characters (or code points), such as `Emoji`. Yet there are many cases in Unicode where properties *of strings* are needed. For example, `Emoji_Flag_Sequence` is really a property of strings, not just of code points. Limiting the terminology to just code points makes the application to high-runner cases like regular expressions unnecessarily awkward. For example, `\p{Emoji}` should be the set of all code points with the `Emoji` property value = `True`, and `\p{Emoji_Flag_Sequence}` should be the set of all strings with the `Emoji_Flag_Sequence` property value = `True`.

Goal:

Having the Unicode Consortium explicitly approve (or reject) this proposal would unblock a Stage 2 ECMAScript feature proposal, and allow it to continue advancing through the standardization process in Technical Committee 39 of Ecma International.

Details:

This document lists all of the important places where text would need to be changed, with specific proposed additional text or rewordings. These are in UAX 44, UTS 18, and UTS 51 and its data files.

There may be some additional places in the Unicode Standard or elsewhere that should be changed as well.

UAX 44 Unicode Character Database

5. Properties

This section documents the Unicode character properties, relating them in detail to the particular UCD data files in which they are specified. For enumerated properties in particular, this section also documents the actual values which those properties can have.

Suggested change:

5. **Properties of characters**

This section documents the Unicode character properties, relating them in detail to the particular UCD data files in which they are specified. For enumerated properties in particular, this section also documents the actual values which those properties can have.

Note that a “property of characters” (e.g. ID_Start) is a property of an individual code point: that is, a function from code points onto some domain of values. A “property of strings” (e.g. Emoji_Keycap_Sequence), on the other hand, is a property of a *sequence* of code points: that is, a function from sequences of code points onto some domain of values.

UTS 51 Unicode Emoji

These characters are in the emoji-sequences.txt file listed under the **type_field** Emoji_Keycap_Sequence

Suggested replacement:

These characters are in the emoji-sequences.txt file listed under the **property** Emoji_Keycap_Sequence.

…and similar for every other use of **type_field** in this document.

ED-20. basic emoji set — The set of emoji code points and emoji presentation sequences listed in the emoji-sequences.txt file [emoji-data] under the **type_field** Basic_Emoji.

Suggested replacement:

ED-20. basic emoji set — The set of emoji code points and emoji presentation sequences listed in the emoji-sequences.txt file [emoji-data] under the **property** Basic_Emoji**_Sequence**.

UTS 18 Unicode Regular Expressions

1.2 Properties

Because Unicode is a large character set, a regular expression engine needs to provide for the recognition of whole categories of characters as well as simply ranges of characters; otherwise the listing of characters becomes impractical and error-prone. This is done by providing syntax for sets of characters based on the Unicode character properties, and allowing them to be mixed with lists and ranges of individual code points.

Suggested replacement:

1.2 Properties

Because Unicode is a large character set, a regular expression engine needs to provide for the recognition of whole categories of characters, ranges of characters**, and categories of strings**; otherwise the listing of characters becomes impractical and error-prone. This is done by providing syntax for sets of characters based on the Unicode character properties, and allowing them to be mixed with lists and ranges of individual code points**; in addition, regular expression engines may provide recognition for the Unicode properties of strings**.

```
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> (":" | "=" | "≠" | "!=" ) VALUE

PROP_SPEC  := <script_or_category_property_value> ("|"
<script_or_category_property_value>)*

PROP_VALUE := <unicode_property_value> ("|" <unicode_property_value>)*
```

The following table shows examples of this extended syntax to match properties: [...]

Suggested addition:

```
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> (":" | "=" | "≠" | "!=" ) VALUE

PROP_SPEC  := <script_or_category_property_value> ("|"
<script_or_category_property_value>)*

PROP_VALUE := <unicode_property_value> ("|" <unicode_property_value>)*
```

Note that the abovementioned `NEGATIVE_SPEC` grammar is not permitted with for properties of strings, e.g. `\P{Emoji_Keycap_Sequence}`.

The following table shows examples of this extended syntax to match properties: [...]

The following table shows examples of this extended syntax to match properties:

[…table entries…]
\p{Whitespace} **anything** that has binary property value Whitespace = True

Suggested addition to the table:

The following table shows examples of this extended syntax to match properties:

[…table entries…]
\p{Whitespace} **any code point** that has binary property value Whitespace = True
\p{Emoji_Keycap_Sequence} any sequence that has the binary property of strings Emoji_Keycap_Sequence = True (matching currently undefined)

---

emoji-sequences.txt

```
# Format:
#   code_point(s) ; type_field ; description # comments
# Fields:
#   code_point(s): one or more code points in hex format, separated by
spaces
#   type_field, one of the following:
#       Basic_Emoji
#       Emoji_Keycap_Sequence
#       Emoji_Flag_Sequence
#       Emoji_Tag_Sequence
#       Emoji_Modifier_Sequence
#     The type_field is a convenience for parsing the emoji sequence
files, and is not intended to be maintained as a property.
```

Suggested replacement:

```
# Format:
#   code_point(s) ; property ; description # comments
# Fields:
#   code_point(s): one or more code points in hex format, separated by
spaces
#   property, one of the following:
#       Basic_Emoji_Sequence
#       Emoji_Keycap_Sequence
#       Emoji_Flag_Sequence
#       Emoji_Tag_Sequence
#       Emoji_Modifier_Sequence
```

Throughout the rest of the data file, apply the same Basic_Emoji → Basic_Emoji_Sequence replacement.

---

emoji-zwj-sequences.txt

```
# Format:
#   code_point(s) ; type_field ; description # comments
#   type_field: Emoji_ZWJ_Sequence
#     The type_field is a convenience for parsing the emoji sequence
files, and is not intended to be maintained as a property.
```

Suggested replacement:

```
# Format:
#   code_point(s) ; property ; description # comments
#   property: Emoji_ZWJ_Sequence
```