L2/19-168

Supporting string properties in regular expressions

by Mathias Bynens (mths@google.com)

Goals:

Having the Unicode Consortium explicitly approve/reject or respond to the following points 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.

1. Get UTC-level resolution on whether to use `\p{…}` or `\q{…}` syntax for accessing string properties (“sequence properties”) in regular expressions.
2. Adopt a stability guarantee stating that any non-string properties may never become string properties in future versions of the Unicode Standard. (Without this guarantee, a backwards-compatibility issue occurs where existing code of the form `\P{Foo}` or `[\p{Foo}]` suddenly starts throwing exceptions.)

Details:

UTS18 defines syntax for accessing various types of Unicode properties in regular expression patterns:

```
\p{Script_Extensions=Greek}      // catalog property
\p{General_Category=Control}     // enumeration property
\p{White_Space}                  // binary property
\p{Numeric_Value=4}              // numeric property
\p{Case_Folding=F}               // string property
\p{Name=BOM}                     // miscellaneous property
```

JavaScript (formally ECMAScript) currently supports various catalog, enumeration, binary, and properties through `\p{…}`. A proposal adds support for string properties:

```
\p{Emoji_Flag_Sequence}
\p{Emoji_Keycap_Sequence}
\p{Emoji_Modifier_Sequence}
\p{Emoji_Tag_Sequence}
\p{Emoji_ZWJ_Sequence}
\p{Basic_Emoji}
```

Unlike other types of properties, string properties cannot be negated using `\P{…}` and cannot occur within character classes. The current proposal therefore bans the use of string properties in those scenarios:

```
\p{Emoji_Keycap_Sequence}        // works
\P{Emoji_Keycap_Sequence}        // throws an exception
[\p{Emoji_Keycap_Sequence}]      // throws an exception
[^\p{Emoji_Keycap_Sequence}]     // throws an exception
```

However, another approach would be to introduce new syntax specifically for string properties, such as `\q{…}`:

```
\q{Emoji_Keycap_Sequence}        // works
\Q{Emoji_Keycap_Sequence}        // throws an exception
[\Q{Emoji_Keycap_Sequence}]      // throws an exception
[^\Q{Emoji_Keycap_Sequence}]     // throws an exception
```

Before advancing this proposal further, TC39 wants explicit guidance from UTC on how to proceed. **Should we use `\p{…}` for string properties, or should we introduce new syntax?**

The case for `\p{…}`

Introducing new syntax comes at a cost for JavaScript developers. In this case, the proposal champion asserts that the cost of adding new syntax for this functionality outweighs the benefits.

The mental model for developers currently is: `\p{…}` refers to a Unicode property. Continuing to use the familiar `\p{…}` syntax for the new string properties proposal means that this **mental model remains unchanged**.

`\P{…}` and `[\p{…}]` throw exceptions for string properties, because that logically follows from that mental model: the exact behavior of `\p{foo}` and `\P{foo}` has always depended on Unicode's definition of `foo`, and continues to do so with this proposal. Developers already have to know the meaning of `foo` to understand how this regular expression pattern behaves today. For example, if `foo` is not a valid Unicode property, it throws an exception.

This is also motivated by UTS18 which uses `\p{…}` even for string properties (RL2.7 explicitly lists `Case_Folding`, for example).

The case for `\q{…}` (or something else)

Proponents of new syntax claim that it would make the distinction between other types of properties (such as binary properties) vs. string properties more clear. A developer could tell that `\Q{foo}` or `[\q{foo}]` would throw just by reading the code, even without knowing how Unicode defines `foo`. However, as explained in the previous section, developers have to know how Unicode defines `foo` anyhow (and if it's a valid Unicode property in the first place) to understand and write such regular expression patterns.

Additionally, UTS18 already defines `\q{foo}` syntax, and it explicitly supports cases we need to ban for string properties: UTS18's `\q{foo}` works within character classes. If we decide to go with new syntax, it'd have to be something other than `\q{foo}` to avoid this conflict with UTS18.

Stability guarantee

Regardless of the outcome of the above, we propose that UTC adopt a stability guarantee stating that any non-string properties may never become string properties in future versions of the Unicode Standard (if such a guarantee does not yet exist).

Without this guarantee, a backwards-compatibility issue occurs where existing code of the form `\P{foo}` or `[\p{foo}]` suddenly starts throwing exceptions if Unicode's definition of `foo` changes from a non-string property to a string property. Note that this back-compat issue would be worse if new syntax like `\q{foo}` is introduced: in that case, even `\p{foo}` would start throwing exceptions.

Supporting string properties in regular expressions

Terminology: string properties

- `Emoji_Flag_Sequence`
- `Emoji_Keycap_Sequence`
- `Emoji_Modifier_Sequence`
- `Emoji_Tag_Sequence`
- `Emoji_ZWJ_Sequence`
- `Basic_Emoji`
- …possibly more in the future

Markus Scherer’s proposal

- **RGI_**Emoji_Flag_Sequence
- Emoji_Keycap_Sequence
- **RGI_**Emoji_Modifier_Sequence
- **RGI_**Emoji_Tag_Sequence
- **RGI_**Emoji_ZWJ_Sequence
- Basic_Emoji
- …possibly more in the future

```
// Existing functionality:
const regexGreekSymbol = /\p{Script_Extensions=Greek}/u;
regexGreekSymbol.test('π');
// → true
```

```
// Existing functionality:
const regexGreekSymbol = /\p{Script_Extensions=Greek}/u;
regexGreekSymbol.test('π');
// → true

// New proposal:
const regexEmojiKeycap = /\p{Emoji_Keycap_Sequence}/u;
regexEmojiKeycap.test('4️⃣'); // '4\uFE0F\u20E3'
// → true
```

```
// Unified syntax with \p{…}
\p{Emoji}                       // works
\P{Emoji}                       // works
\p{Emoji_Keycap_Sequence}       // works
\P{Emoji_Keycap_Sequence}       // throws an exception
[\p{Emoji_Keycap_Sequence}]     // throws an exception
[^\p{Emoji_Keycap_Sequence}]    // throws an exception
```

```
// Disunified syntax with \q{…}
\p{Emoji}                       // works
\P{Emoji}                       // works
\q{Emoji}                       // throws an exception
\p{Emoji_Keycap_Sequence}       // throws an exception
\P{Emoji_Keycap_Sequence}       // throws an exception
\q{Emoji_Keycap_Sequence}       // works
\Q{Emoji_Keycap_Sequence}       // throws an exception
[\q{Emoji_Keycap_Sequence}]     // throws an exception
[^\q{Emoji_Keycap_Sequence}]    // throws an exception
```

```
// mental model for developers:

\p{Foo}

// refers to the Unicode property Foo
```

Stability guarantee

Proposal: adopt a stability guarantee stating that any non-string properties may never become string properties in future versions of the Unicode Standard (if such a guarantee does not yet exist).

Without this guarantee, a backwards-compatibility issue occurs where existing code of the form `\P{foo}` or `[\p{foo}]` suddenly starts throwing exceptions if Unicode's definition of foo changes from a non-string property to a string property. Note that this back-compat issue would be worse if new syntax like `\q{foo}` is introduced: in that case, even `\p{foo}` would start throwing exceptions.