Full Unicode in ECMAScript

Norbert Lindenberg
Please ask questions!
# Encoding Unicode

<table>
<thead>
<tr>
<th>Character</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code point</td>
<td>U+0061</td>
<td>U+03B1</td>
<td>U+5409</td>
<td>U+20BB7</td>
</tr>
<tr>
<td>UTF-32</td>
<td>00000061</td>
<td>000003B1</td>
<td>00005409</td>
<td>00020BB7</td>
</tr>
<tr>
<td>UTF-16</td>
<td>0061</td>
<td>03B1</td>
<td>5409</td>
<td>D842•DFB7</td>
</tr>
<tr>
<td>UCS-2</td>
<td>0061</td>
<td>03B1</td>
<td>5409</td>
<td>—</td>
</tr>
<tr>
<td>UTF-8</td>
<td>61</td>
<td>CE•B1</td>
<td>E5•90•89</td>
<td>F0•A0•AE•87</td>
</tr>
</tbody>
</table>
# Encoding Unicode

<table>
<thead>
<tr>
<th>Character</th>
<th>a</th>
<th>α</th>
<th>吉</th>
<th>吉</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code point</td>
<td>U+0061</td>
<td>U+03B1</td>
<td>U+5409</td>
<td>U+20BB7</td>
</tr>
<tr>
<td>UTF-32</td>
<td>00000061</td>
<td>000003B1</td>
<td>00005409</td>
<td>00020BB7</td>
</tr>
<tr>
<td>UTF-16</td>
<td>0061</td>
<td>03B1</td>
<td>5409</td>
<td>D842•DFB7</td>
</tr>
<tr>
<td>UCS-2</td>
<td>0061</td>
<td>03B1</td>
<td>5409</td>
<td>—</td>
</tr>
<tr>
<td>UTF-8</td>
<td>61</td>
<td>CE•B1</td>
<td>E5•90•89</td>
<td>F0•A0•AE•87</td>
</tr>
</tbody>
</table>
Today: UCS-2 or UTF-16?

UCS-2:
- Regular expressions
- String comparison
- Case conversion

UTF-16:
- Source text conversion
- URI handling
Today: UCS-2 or UTF-16?

UCS-2:
- Regular expressions
- String comparison
- Case conversion

UTF-16:
- Source text conversion
- URI handling
- DOM, text input, text rendering, XMLHttpRequest, libraries, apps
Full Unicode?

- One code point === one string element?
  - UTF-32

- All Unicode characters supported, somehow?
  - UTF-32 or UTF-16
UTF-32

+ Easy to understand, easy to use
  – Breaks code that assumes UTF-16
  – Breaks code that transmits index information without translation
  – Unclear how to interpret \uD842\uDFB7
UTF-32/16 switch

+ Locally easy to understand, easy to use

+ Compatibility box for old code
  - Breaks code that gets run with wrong setting; requires libraries to support both
  - Breaks code that transmits index information without translation
  - Unclear whether \uD842\uDFB7 should be allowed
UTF-16

+ Compatible with existing code
+ Compatible with index transmission
+ Code-point based regex, string functions, string iteration possible
  - Requires low-level developers to think in both code units and code points
Priorities

1. Code-point based regular expressions
2. Supplementary characters in functions
3. Supplementary characters everywhere
4. One code point === one string element
5. Code-point based string accessors
6. Code point escapes \u{20BB7}
Proposal

1. Code-point based regular expressions
2. Supplementary characters in functions
3. Supplementary characters everywhere
4. One code point === one string element
5. Code-point based string accessors
6. Code point escapes \u{20BB7}
Basics

- Define code unit, code point
- Define interpretation of code unit sequence as code point sequence
- Well-formedness not required
Regular expressions

• Patterns and input interpreted as code points
  • /.\ matches code point, not code unit
• Supplementaries as range limits
• Case insensitive matching for all
• Workaround for workarounds
• Some compatibility issues – /u needed?
/u – little red switch?

- Unicode code point semantics
- Unicode based \d\D\w\W\b\B
- Unicode case folding
- Remove some/all identity escapes to allow future extensions: \p, \X, \N
- Don’t match web reality?
Other text processing

- Case conversion: toLowerCase & Co.
- Any future functions
- Not: relational comparison for strings
Complete Unicode

• Unicode 5.1
• No more UCS-2
• Code point based identifiers: ♫, ☛
• Clean up specification
Code point access

- String.fromCharCode([cp0 [, cp1 [, ...]]])
- String.prototype.codePointAt(pos)
- String.prototype.[iterator]
Code point escape

- “\u{20BB7}” === “𠮷” =?= “\uD842\uDFB7”
- 1-6 hex digits; value 0–0x10FFFF
- Exclude 0xD800–0xDFFF?
- Use in identifier, string literal, regex literal; not in JSON
- Interpretation context-sensitive, as for \uxxxx