Unicode Support in ECMAScript Edition 4

Markus W. Scherer
2003-feb-07

Since ECMAScript edition 3 was published, the Unicode standard has added features, fixed bugs, and improved specifications. This document is a proposal to update Unicode-related chapters in ECMAScript edition 4 accordingly.

References to chapters refer to the current, draft of edition 4 (draft from 2003-jan-13).

Unicode Version

Chapter 6 requires that ECMAScript implementations use Unicode version 2.1 or higher.

Proposal: To require implementations to use Unicode version 3.0 or higher.

The higher the Unicode baseline version, the more characters can be used reliably in identifiers etc. However, while the current version of Unicode is 3.2, it is expected that some ECMAScript implementations are written in Java, where only Unicode 3.0 is supported at this time. Requiring a higher version of Unicode would put an additional burden on such implementations. Unicode 3.0 was published in 1999.

Ignorable Characters

Chapter 6.1 specifies that implementations strip ignorable characters from the script text before any further parsing. Ignorable characters are defined as those with a Unicode general category value of Cf (format controls).

Proposal: To change the set of ignorables from Cf to Default_Ignorable_Code_Point, which is a superset of Cf.

The Default_Ignorable_Code_Point property was introduced in Unicode 3.2. In addition to Cf characters, it contains further characters with similar control functions. It also covers ranges of unassigned code points that are reserved for future control characters. In particular, in addition to Cf characters, it lists:

- Variation selectors: Characters that are invisible by themselves but select a particular, predefined glyph for the preceding character if the display system supports it. They are given a general category value of Mn (non-spacing combining mark) to form a logical unit with the "real" character in various Unicode algorithms.
- ISO control codes: ISO control codes other than those commonly used as white space (general category Cc, excluding characters with the White_Space property).
- Surrogate code points: Surrogate code points (Cs) should not be exchanged in text at all. In 16-bit Unicode text, they appear as lead surrogates without following trail surrogates, or trail surrogates without preceding lead surrogates. (Surrogate *pairs* in 16-bit Unicode text encode supplementary code points.)
- Future ignorables: Three ranges of Unicode code points are reserved for current and future "control" characters. This improves the stability of Unicode text handling because most ignorables in future Unicode versions are already recognized as such.

For ECMAScript implementations that use Unicode versions before 3.2, the Default_Ignorable_Code_Point property should be implemented as in Unicode 3.2:

```
# Derived Property: Default_Ignorable_Code_Point
#  Generated from <2060..206F, FFF0..FFFB, E0000..E0FFF>
#    + Other_Default_Ignorable_Code_Point + (Cf + Cc + Cs - White_Space)

0000..0008    ; Default_Ignorable_Code_Point # Cc   [9] <control>..<control>
000E..001F    ; Default_Ignorable_Code_Point # Cc  [18] <control>..<control>
007F..0084    ; Default_Ignorable_Code_Point # Cc   [6] <control>..<control>
0086..009F    ; Default_Ignorable_Code_Point # Cc  [26] <control>..<control>
06DD          ; Default_Ignorable_Code_Point # Cf       ARABIC END OF AYAH
070F          ; Default_Ignorable_Code_Point # Cf       SYRIAC ABBREVIATION MARK
180B..180D    ; Default_Ignorable_Code_Point # Mn   [3] MONGOLIAN FREE VARIATION SELECTOR ONE..MONGOLIAN FREE VARIATION SELECTOR THREE
180E          ; Default_Ignorable_Code_Point # Cf       MONGOLIAN VOWEL SEPARATOR
200C..200F    ; Default_Ignorable_Code_Point # Cf   [4] ZERO WIDTH NON-JOINER..RIGHT-TO-LEFT MARK
202A..202E    ; Default_Ignorable_Code_Point # Cf   [5] LEFT-TO-RIGHT EMBEDDING..RIGHT-TO-LEFT OVERRIDE
2060..2063    ; Default_Ignorable_Code_Point # Cf   [4] WORD JOINER..INVISIBLE SEPARATOR
2064..2069    ; Default_Ignorable_Code_Point # Cn   [6]
206A..206F    ; Default_Ignorable_Code_Point # Cf   [6] INHIBIT SYMMETRIC SWAPPING..NOMINAL DIGIT SHAPES
D800..DFFF    ; Default_Ignorable_Code_Point # Cs [2048]
FE00..FE0F    ; Default_Ignorable_Code_Point # Mn  [16] VARIATION SELECTOR-1..VARIATION SELECTOR-16
FEFF          ; Default_Ignorable_Code_Point # Cf       ZERO WIDTH NO-BREAK SPACE
FFF0..FFF8    ; Default_Ignorable_Code_Point # Cn   [9]
FFF9..FFFB    ; Default_Ignorable_Code_Point # Cf   [3] INTERLINEAR ANNOTATION ANCHOR..INTERLINEAR ANNOTATION TERMINATOR
1D173..1D17A  ; Default_Ignorable_Code_Point # Cf   [8] MUSICAL SYMBOL BEGIN BEAM..MUSICAL SYMBOL END PHRASE
E0000         ; Default_Ignorable_Code_Point # Cn
E0001         ; Default_Ignorable_Code_Point # Cf       LANGUAGE TAG
E0002..E001F  ; Default_Ignorable_Code_Point # Cn  [30]
E0020..E007F  ; Default_Ignorable_Code_Point # Cf  [96] TAG SPACE..CANCEL TAG
E0080..E0FFF  ; Default_Ignorable_Code_Point # Cn [3968]

# Total code points: 6271
```

Cn is the general category value for unassigned code points.

Note that the above list segments contiguous ranges by general categories as usual for lists of Unicode properties. This is purely for documentation.

Line Breaks

Chapter 7.3 defines a small number of characters as LineTerminator characters.

Proposal: To add the ISO Control U+0085 Next Line (NEL) to the list of LineTerminator characters.

ECMAScript already lists several Unicode and ISO 6429 controls that are commonly used as line break characters, except for U+0085 Next Line. See UAX #13: Unicode Newline Guidelines and XML 1.1.

Note that in edition 3 there was at least one other place where these characters were part of a production: Edition 3 chapter 9.3.1 ToNumber defines StrWhiteSpaceChar with what seems to be a union of WhiteSpace and LineBreak. (In edition 4, chapter 18.7.1 appears to be reserved for this.) I propose that such definitions be replaced with references to common definitions. This would make the standard more maintainable and avoid errors. Alternatively, each such definition should be updated in parallel.

Identifiers

Chapter 7.5 defines identifier characters based on Unicode general categories, equivalent to the Unicode ID_Start and ID_Continue properties. ECMAScript also requires that at least the *Unicode 3.0* ID_Start and ID_Continue characters are recognized. This is for stability: Very occasionally, Unicode changes general category assignments to fix problematic cases, but this can cause some characters to be suitable for identifiers in some earlier version of Unicode but not the current, later one.

Proposal: To include the new Unicode property ID_Start_Exceptions into the definition of ECMAScript identifiers.

The ID_Start_Exceptions property is being created so that one can implement stable identifier rules effectively. It lists all characters that were ID_Start in some Unicode version (2.1 or higher) but are not in the current Unicode version.

For ECMAScript implementations that use Unicode versions before 4.0, the ID_Start_Exceptions property should be implemented as in Unicode 4.0 (PropList.txt, showing pre-beta data here):

```
2118 SCRIPT CAPITAL P
212E ESTIMATED SYMBOL
309B KATAKANA-HIRAGANA VOICED SOUND MARK
309C KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK
```

Note that the Unicode 4.0 character database is expected to be final around April of 2003, in time for inclusion of this property into ECMAScript edition 4.

Unicode Escapes

Chapter 7.8 defines HexEscape sequences (with alternatives \xhh and \uhhhh) for string literals. They are limited to code point values for the BMP (<=U+FFFF). When a developer needs to represent supplementary code points, they have to manually calculate the two UTF-16 surrogate code units and put the two \uhhhh sequences into the source code. This is tedious and error-prone. For example, U+20001 would have to be written as `\ud840\udc01`. (Java does not yet have such an escape sequence syntax either, but several other language standards do.)

Proposal: To add escape sequences to ECMAScript that can directly represent supplementary Unicode code points (<=U+10FFFF).

Note that it is possible, but not necessary, to restrict such new escape sequences to only supplementary code points (10000..10FFFF but not <10000). ECMAScript permits \uhhhh for characters where \xhh would suffice, and other standards do not appear to have such restrictions.

There are several reasonable syntaxes for such escape sequences:

- \Uhhhhhhhh — Fixed-length format like \uhhhh but with 8 hex digits. Simple, but only 6 digits are ever needed. This is used in recent C/C++/Python standards. (See http://www.jorendorff.com/articles/unicode/python.html)
  Example: `\U00020001`
- \x{hh...h} — Variable-length format with 1..6 hex digits. We could allow an arbitrary number of leading zeroes for simple syntax. This is used in Perl. (See http://www.xav.com/perl/lib/Pod/perlunicode.html)
  Example: `\x{20001}`
- \u{hh...h} — Variable-length format with 1..6 hex digits, same as the previous one but with \u instead of \x, which seems more natural for Unicode and for a language whose users already use \u for BMP characters.
  Example: `\u{20001}`

Of course, more than one of these could be allowed.

(HTML and XML use variable-width syntaxes `&#dd...d;` and `&#xhh...h;`)

Semantically, an escape sequence for a supplementary code point would add two "characters" (ECMAScript terminology; two invocations of codeToCharacter()) to the string or regular expression etc.

Similarly to the comment about LineBreak above, there are several ECMAScript edition 3 productions that define escape sequences. Edition 4 appears to already re-use a single definition.

Terminology

The Unicode standard has modified and clarified some of its terminology. For example, "Unicode values" and "scalar values" should be changed to "Unicode code points", the use of "surrogate" vs. "supplementary character" needs to be cleaned up, etc.

Proposal: To update Unicode-related text in the ECMAScript standard to reflect changes in Unicode terminology. This would be best done editing the chapters with change bars for review.