I18N Support in ECMAScript Edition 4

Markus W. Scherer
2003-feb-21

This document is a proposal to update Internationalization-related chapters in ECMAScript edition 4.

Thanks to Mark Davis and Richard Gillam for an initial review and helpful comments.

String Construction from Supplementary Characters

Edition 3 section 15.5.3.2 defines String.fromCharCode() which takes zero, one, or more 16-bit Unicode code unit values and creates the corresponding string. It does not directly handle supplementary code points. This is inconvenient, see the discussion of \u in my earlier Unicode support proposal for ECMAScript Edition 4 (es4-unicode.html).

Proposal: To extend String.fromCharCode() to accept supplementary Unicode code points. Instead of ToUInt16, each argument would be converted with ToInt32 (or ToUInt32). Values 0..ffff are processed as before. Values 10000..10ffff yield a pair of surrogate code units. Other values throw an exception.

For example, instead of String.fromCharCode(0x61, 0xd840, 0xdc01) one could then simply write String.fromCharCode(0x61, 0x20001) and get the same string.

String Comparisons

Edition 3 section 15.5.4.9 String.prototype.localeCompare() has a very abbreviated explanation of canonical equivalence, and one part requires canonical equivalence while another only recommends it.

Proposal: To update the text for this function by largely referring to relevant sections of the Unicode Standard, without expanding the required semantics of the function. Specifically (changes with ~~strikethrough~~ and *italics*):

In the second NOTE make the following changes: "This function is intended to rely on whatever language-sensitive comparison functionality is available to the ECMAScript environment from the host environment, and to compare according to the ruls of the host environment's current locale. It is ~~strongly recommended~~ *required* that this function treat strings that are canonically equivalent according to the Unicode standard as identical ~~(in other words, compare the strings as if they had both been converted to Normalised Form C or D first)~~. It is also recommended that this function not honour Unicode compatibility equivalences or decompositions. *For a definition and discussion of canonical equivalence see the Unicode Standard, chapters 2 and 3, as well as UAX #15 Unicode Normalization Forms and UTN #5 Canonical Equivalence in Applications. See also UTS #10 Unicode Collation Algorithm."*

Change the following paragraph: "If no language-sensitive comparison at all is available from the host environment, this function may perform a ~~bitwise~~ *canonical equivalence* comparison."

Message Formatting

Internationalized programming requires message formatting where message values are inserted at grammatically appropriate positions. Depending on the language, the order of such value inserts in a message sentence may change.

Proposal: To add a function like String.formatMessage(format, valueMap) that supports variable-position

inserts.

The valueMap parameter is an ECMAScript object (an array). It can be null if the format does not specify any inserts. The format parameter is a string with insertion parameters marked with braces, such as {name}. While formatting, each insertion parameter is replaced by the toString() value of the property of valueMap whose property name matches the insertion name. Example (not tested):

```
var myFormat = "The work order {workOrder} for {name} was processed on {date}";
// later in the program
var valueMap = new Object();
valueMap["name"] = "Mr. Smith";
valueMap["workOrder"] = 592473;
valueMap["date"] = Date.now();
var fmtOutput = String.formatMessage(myFormat, valueMap);
```

The format is based on a Java syntax for MessageFormat, simplified somewhat. Inside the braces and after the name, a comma and following text is ignored for future expandability (allowing the addition of format strings such as "###,##0.00" as in Java sometime in the future). If real curly braces are required in the string, they can be quoted. Single quotes around text inhibits parsing of insert braces inside the quotation. A pair of single quotes results in just a single quote character.

Date/Time Formatting

ECMAScript provides the Date object for parsing and formatting of dates with an unknown format. For validation and localized UI elements, it is required to know the format and elements of the date/time format patterns.

For example, HTML forms for date/time input often use separate fields for the year, month, day, etc. It is necessary to determine the order of those fields programmatically in order to generate such a form appropriately for the browser's locale. For selection UI elements (dropdown controls), the names of the fields are required as well to populate the selection choices.

Proposal: To add functions to the Date object to get each month and day name ("März", "Dienstag"), to get localized AM/PM names, and to get date, time, and date+time format patterns as used by the Date object (e.g., an array like ["Y", "M", "D", "h", "m"]).

There are various ways to do this. For example: getDayName(index), getMonthName(index), getAMName(), getPMName(), getDatePattern(), getTimePattern(), getDateTimePattern().

The getXYZPattern() functions could return array objects corresponding to the order and types of fields used for formatting and parsing date/time strings. Each array element could be

- A single-letter string like "Y" for the year field (using Java date/time pattern characters).
- A string like "year" for the year field (using Java date/time pattern field names).
- A number like 1 for the year field (using Java date/time pattern field indexes).

Locale-Specific Functions

Edition 3 suggests for a number of functions that a locale ID parameter may be added in the future. For example, String.prototype.localeCompare(that)'s second parameter position.

Proposal: To review such functions and decide if a locale ID parameter or similar should be added for Edition 4.

Regular Expressions

This is more supplementary character support. Edition 3 defines regular expressions in terms of Unicode code units, not Unicode code points. At its core, the CharSet productions are limited to 16-bit values.

Proposal: To extend Regular Expressions (CharSet etc.) to support supplementary code points.

In the future, ECMAScript Regular Expressions should be upgraded to the level of Unicode support offered by Perl. See also the discussion of Basic Unicode Support (Level 1) in UTR #18 Unicode Regular Expression Guidelines.