Notes of the: **Meeting of Ecma TC39 ad hoc on Internationalization**

held on: **17 August 2011**

ECMAScript i18n : August 17, 2011 @Google

Attendees: Allen (Mozilla), Gendalf (Moz), Eric (Microsoft), Norbert (self), Richard (az), Cira (Google), Jungshik (Google), John Tamplin (Google), Mark Davis (Google)

Place: Google

Date: August 17, 2011

Allen's comments: http://goo.gl/BtEXp

Allen: In JS, a lot of objects are mirrorable, deletable, modifiable.

Cira: 'derive'

Allen: 'options' : do we need

Allen: ctor accepting multiple types. Overloading arguments to make a semantic distinction requires the exact algorithm to be defined.

1. e.g. what properties have to be present for an object to be treated as 'array' : length
2. even a String object looks like an array.
3. JS does not throw an error for a 'wrong' argument type. e.g. if 'string' is required, an argument is converted to 'String' (calling toString of an object). Same is true of 'Number'
4. ECMAScript 4 and 5 may have some differences in 'mechanism' for "distinguishing" types. Need to walk along a narrow path
5. toNumber, toString etc
6. ctor overloading based on argument type is arguably not a good JS style.
7. alternative: have a base ctor and add 2 factory methods accepting different types (options, an array of language tags)

NF.forLanguage(lang1, lang2, lang3....)

NF.forOptions(option)

NF.forLangList(array of langIDs)

John: an alternative is to use multiple strings (language tags) instead of an array of strings (lang tags)

Cira: NF(opt, locale1, locale2,....) : locale is not a part of options, but 'opt' is mandatory

Everybody: how often is the case we need to specify ‘options’?

Everybody: Let’s take an concret class to nail down the way ctro uses argument. Starting with collator

Everybody: strawman : collator spec.

Collator needs an extensible set of options. Currently, it has 3 (numeric, ignorePunctuation, sensitivity), but others can be added.

Mark: immutability makes it necessary to have extensible options

Allen: there are two kinds of arguments: locale(s) and object-specific options

John/Allen: need a way to filter out and pre-determine a locale to use ‘globally’

Mark:

1. user’s locale preference
2. locales supported by apps
3. locales supported by each of services
4. fallback

Allen: compute once what locale to pass to service ctor

John: user’s preferred locales are resolved/intersected with app’s supported locales (#1 and #2 in Mark’s list)

John: how about providing a building block to resolve/intersect multiple lists of locales

Everybody: make simple things simple

John: resurrect LocaleInfo ?

Mark: cleaner to have Locale object with a set of convenience methods for resolving , validation, …..

Allen: other libraries can use such an object

Richard: what if a raw string is passed to our service object?

Mark: locale1 - set #1, locale 2 - set #2. Locale.intersect(locale1, locale2)

John: inconsistency between services is not desirable to app developers

Everybody: LocaleList : takes an array of string locale ids. (ordered list)

Richard: a utility method for converting HTTP A-L header values to an input argument for LocaleList?

Jungshik: is it useful for the client-side script?

Gendalf: what fallback mechansim?

Eric: separate object for Region?

Mark: no. region will be a separate option for specific services

Everybdoy: BCP47, '-u' extension

Cira: g11n vs globalization

Allen: have to prepare for g11 to become a module.

Cira: will there be a conflict with an ES6 module?

MarkS/Allen: in anticipation of g11n becoming a module, do not allow extensible properties in g11n

MarkS/Allen: should g11n be writable in light of it being migrated to a module?

MarkS/Allen: i18n vs g11n?

Eric: should filter require accepting only LocaleList ? or, should it accept an array of strings?

Gendalf/others: behavior of LocaleList is different from an array of strings.

Mark: matching behavior of 'filter'

1. myLocale : fr   otherLocale: fr-CA
2. myLocale: fr-FR   otherLocale: fr-CA
3. myLocale: fr-CA,   otherLocale: fr
4. myLocale: fr   otherLocale: fr

John: add a 2nd param for 'strictness' ?

Mark/others: we need a more flexible filtering behavior

Conclusion

1. LocaleList (array of locale ids) : elements of arrays should be convertible to string.

1. a generic array object. (indexed access, length)
2. property is enumerable
3. read-only, non-configurable, extensible

myLocale[0] = 3 : not allowed

myLocale.foo = 3; allowed

2. At least one instance method

1. LocaleList.prototype.filter(otherLocaleList)
2. input argument (otherLocaleList) is treated as unordered
3. ‘this’ is treated as ordered
4. a new instance of LocaleList is returned (the result of filtering)

3. Empty LocaleList is treated as default locale

1. Filter with disjoint sets can produce empty LocaleList
2. Empty LocaleList constructor creates an object with default locale
3. LocaleList constructor with empty array constructs empty LocaleList

4. LocaleList

1. ill-formed langID in ctor throws an exception (ie, not BCP47 compliant)
2. duplicated elements are merged
3. en-US is different from en-Latn-US and en
4. result will be in canonical BCP 47 format

x = new LocaleList("En-Us", "en-us"])

x[0] = "en-US",   x.length = 1

Collation

1. numeric : true/false or absent
2. ignorePunctuation: true/false or absent
3. sensitivity : "base", "accent", "case", "variants", or absent
4. "fred" is equivalent to absent/missing
5. John: make it discoverable ?
6. Richard:

1. accept whatever value and let caller query

7. Eric: 1. search vs sort 2. sensitivity (MS can support)
8. John: add 'two' immutable constants for search and sort
9. Richard/Jungshik: add 'usage' key with values of 'search' and 'sort' (and absent)
10. if nothing is specified, default to sort-locale-default
11. locale IDs vs options : options overide localeIDs (-u- extension)
12. Mark: 'numeric' is very tough to define : thousand separators, non-ASCII digits, floating points?
13. Richard: do we have to define some 'baseline' implementation spec?
14. Allen:
15. Eric: implementation is free to honor or disregard what's specified in locale ids ?
16. Richard/Nobert/Allen/Jungshik: if an implementation is capable of 'numeric', why would not any implementation want to honor 'u-co-kn' ?

default	V- - -
language tag	- VV-

Options	- - VV
Resolved options	VVVV
Resolved LT	VVVV

1. u-co-kn=no => should show up in ROpt and RLT

DateTimeFormat

1. Add "FirstDayOfWeek"
2. -u-ca-buddihst
3. Eric: Arabic calendar widget can't be built without algorithms. So, getXXX is not so useful
4. Richard: don't feel like having getXXX in DateTimeFormat ...

Conclusion

5. Add 'calendar' option to DateTimeFormat
6. Remove getFoo's
7. Remove styles

{

hour: “abbreviated”,

month: “long”,

year: “abbreviated”

}

List out invalid combinations in skeleton

Remove skeleton property and make all items top level in options

NumberFormat

Cira: Currently it has patterns and styles

Nobert proposal: http://norbertlindenberg.com/ecmascript/internationalization-formats.html

Remove skeleton and pattern

Add what Norbert proposed : multiple attributes at the top level in options

toLocaleString for Number and Date

1. What to do about it?
2. Allen: toLocaleString is extensible (can have a 2nd param)
3. Allen/Richard: less intimidating than g11n / i18n such as Format, Presentation, UserInterface, Text

Derive method

1. Allen:
2. Cira: do we need derive?
3. Cira: do we want to move resolved options to the top level?
4. Allen: is inclined to keep them bundled up as they're now
5. Allen: can make 'resolvedSettings' an accessor

Resolution

1. Get rid of derive
2. make 'resolveSettings' an accessor

Spec-writing

1. Nobert: will help with spec ...
2. can't use JS code. have to use pseudo-code
3. Can meet September deadline?
4. Let's use Google Docs for collaboration
5.

Implementation

1. Mozilla : considering ICU (BootToGecko). Likely to have sth. up before Eof 2011
2. IE: Peter said that MS has a team...
3. JSCore: Amazon's implementation : once spec settles down, it will be relatively quick, but upstreaming it to JSCore takes more work
4. Google: ?