Modernizing Internationalization in Gecko and SpiderMonkey

Daniel Minor
SpiderMonkey / Mozilla
Overview

- What is internationalization?
- Deep Dive: Text Segmentation
- Localization in the browser
- How is all of this implemented?
- Experimenting with ICU4X in Firefox
- ICU4X Text Segmentation
What is Internationalization?
What is Internationalization?

• Part of a group of related ideas
  ○ Internationalization (i18n)
  ○ Translation
  ○ Localization (l10n)

• Let’s talk about translation first
### Translation

<table>
<thead>
<tr>
<th>Language and Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fonts and Colors</strong></td>
</tr>
<tr>
<td>Default font: Default (Times)</td>
</tr>
<tr>
<td>Size: 16</td>
</tr>
<tr>
<td>Advanced...</td>
</tr>
<tr>
<td>Colores...</td>
</tr>
<tr>
<td><strong>Zoom</strong></td>
</tr>
<tr>
<td>Default zoom: 100%</td>
</tr>
<tr>
<td>Zoom text only</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language and Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Typefaces and Colors</strong></td>
</tr>
<tr>
<td>Predeterminado: Predeterminado</td>
</tr>
<tr>
<td>Tamaño: 16</td>
</tr>
<tr>
<td>Avanzadas...</td>
</tr>
<tr>
<td>Colores...</td>
</tr>
<tr>
<td><strong>Zoom</strong></td>
</tr>
<tr>
<td>Ampliación predeterminado: 100%</td>
</tr>
<tr>
<td>Zoom text only</td>
</tr>
</tbody>
</table>
Localization

- Translation + Cultural Adaptation
- Appropriate use of colours, symbols, images, etc.
- E.g. Bug 615495 - Hide nurse when locale is 'ja'
  - Around 12 years ago Mozilla support site redesign featured a cartoon cat nurse
  - In Japanese culture, this had unintended erotic connotations
Internationalization

- Internationalization consists of all the things that can be done by computer
  - E.g. Dates, times, currency formatting
- Internationalization enables localization
  - Less work for translators
- Data driven
  - CLDR - Common Locale Data Repository
  - XML and json
  - Unicode Technical Report #35 defines the schema and interpretation of this data
# Internationalization Examples

<table>
<thead>
<tr>
<th>Category</th>
<th>Example 1</th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbering systems</td>
<td>1, 2, 3, 4, ...</td>
<td>٠، ١، ٢، ٣، ٤</td>
</tr>
<tr>
<td>Number groupings</td>
<td>133,000.00</td>
<td>1,33,000.00</td>
</tr>
<tr>
<td>Date &amp; Time</td>
<td>June 5, 2022 at 4:46:55 PM</td>
<td>2022年6月5日 16:46:41</td>
</tr>
<tr>
<td>Calendar systems</td>
<td>Tuesday, January 12, 2021</td>
<td>28 Tevet 5781</td>
</tr>
<tr>
<td>Text segmentation</td>
<td>Lorem ipsum dolor sit amet,</td>
<td>ญเพราะการจะเป็นอิสระได้จะต้องมีกำลังที่มากกว่าแข็งแกร่งกว่า</td>
</tr>
<tr>
<td></td>
<td>consectetur adipiscing elit.</td>
<td></td>
</tr>
<tr>
<td>Directionality</td>
<td>Liked by John and two more</td>
<td>Liked by الويب and two more</td>
</tr>
<tr>
<td></td>
<td>people.</td>
<td>people.</td>
</tr>
<tr>
<td>Lists</td>
<td>Mary, Nick and Liam</td>
<td>Mary, Nick มะกันและ Liam</td>
</tr>
<tr>
<td>Plural Rules</td>
<td>one, other</td>
<td>zero, one, two, few, many, other</td>
</tr>
<tr>
<td>Currencies</td>
<td>$123.45</td>
<td>123.45 $US, $۳۲۱٫۵۴</td>
</tr>
</tbody>
</table>
ECMA-402: Internationalization in JavaScript

- Set of APIs that implement internationalization in JavaScript
- Create a formatter by specifying a locale and some options
- And the formatter provides a format() method to do the work

```javascript
const number = 123456.789;

new Intl.NumberFormat('de-DE', { style: 'currency', currency: 'EUR' }).format(number);
// expected output: "123.456,79 €"

// the Japanese yen doesn't use a minor unit
new Intl.NumberFormat('ja-JP', { style: 'currency', currency: 'JPY' }).format(number);
// expected output: "¥123,457"

// limit to three significant digits
new Intl.NumberFormat('en-IN', { maximumSignificantDigits: 3 }).format(number);
// expected output: "1,23,000"
```
Why Localization and Internationalization?

● For Mozilla, it’s part of our manifesto, “The internet is a global public resource that must remain open and accessible”

● An English only web is neither open nor accessible
  ○ nor Chinese, Spanish, Arabic, etc.
Why Localization and Internationalization?

- Ensure that people are able to access the web in their own languages
  - Localize the browser
- Provide people with the tools they need to localize the web
  - ECMA-402 support in JavaScript
Deep Dive: Text Segmentation
Text Segmentation

- The process of chunking text into meaningful units
  - On character boundaries
  - On word boundaries
    - E.g. jump to the next word or sentence in a text editor
  - On line boundaries
    - E.g. for word wrapping text in a column
Grapheme Breaking: Code Points vs. Graphemes

- Graphemes are what are rendered to the screen
  - 👍 U+1F44D THUMBS UP SIGN
- Code points encode characters
- Not necessarily a 1:1 mapping from code points to graphemes
  - 👍 U+1F44D THUMBS UP SIGN
  - 🅽 U+1F3FE EMOJI MODIFIER FITZPATRICK TYPE-5
Segmentation - Grapheme breaking

```javascript
var string = "thumbs 👍 up"
// 'thumbs 👍 up'

[s...s.segment(string)].map(s => s.segment)
// (11) ['t', 'h', 'u', 'm', 'b', 's', ' ', '👍', ' ', 'u', 'p']

[s...string]
// (12) ['t', 'h', 'u', 'm', 'b', 's', ' ', '👍', ' ', 'u', 'p']
```
Segmentation - Word breaking

- Break on word boundaries
- Complexity depends upon the language
  - A lot easier when there’s spaces between words
  - More complicated for Asian languages
Es tan corto el amor, y es tan largo el olvido.
Word Breaking - Spanish

Es|tan|corto|el|amor,|y|es|tan|largo|el|olvido.
古池や 蛙飛び込む 水の音
古池や 蛙飛び込む 水の音
Intl.Segmenter

- Stage 4 Proposal for ECMA-402
  - Already implemented in Chromium and Safari
- Use case: implementing text editors in JavaScript, etc.
- Current proposal includes grapheme, word and sentence breaking
- But not line breaking:
  - Line breaking is part of a v2 of the proposal
  - To do it properly, need information on text size and position, not just break locations
Intl.Segmenter Example

```javascript
> // Create a locale-specific word segmenter
let segmenter = new Intl.Segmenter("ja", {granularity: "word"});
```
> // Create a locale-specific word segmenter
> let segmenter = new Intl.Segmenter("ja", {granularity: "word"});

> // Use it to get an iterator for a string
> let input = "古池や 蛙飛び込む 水の音";
> let segments = segmenter.segment(input);
Intl.Segmenter Example

```javascript
// Create a locale-specific word segmenter
let segmenter = new Intl.Segmenter("ja", {granularity: "word"});

// Use it to get an iterator for a string
let input = "古池や 蛙飛び込む 水の音";
let segments = segmenter.segment(input);

// Use that for segmentation!
for (let {segment, index, isWordLike} of segments) {
  console.log("segment at code units [%d, %d]: «%s»%s",
               index, index + segment.length,
               segment,
               isWordLike ? " (word-like)" : "");
}
```
Intl.Segmenter Example

```js
// Create a locale-specific word segmenter
let segmenter = new Intl.Segmenter("ja", {granularity: "word"});

// Use it to get an iterator for a string
let input = "古池や 蛙飛び込む 水の音";
let segments = segmenter.segment(input);

// Use that for segmentation!
for (let {segment, index, isWordLike} of segments) {
  console.log("segment at code units [%d, %d]: «%s»%s",
               index, index + segment.length,
               segment,
               isWordLike ? " (word-like)" : "")
}
```

```plaintext
segment at code units [0, 2]: «古池» (word-like)
segment at code units [2, 3]: «や» (word-like)
segment at code units [3, 4]: «  »
segment at code units [4, 5]: «蛙» (word-like)
segment at code units [5, 9]: «飛び込む» (word-like)
segment at code units [9, 10]: «  »
segment at code units [10, 11]: «水» (word-like)
segment at code units [11, 12]: «の» (word-like)
segment at code units [12, 13]: «音» (word-like)
```

< undefined
Localization in the Browser
Localization in the Browser

- ECMA-402 allows developers to use JavaScript to localize the web
- But the browser is also a product that needs to be localized!
We ship 133 Locales in Firefox!

<table>
<thead>
<tr>
<th>Acehnese</th>
<th>Bosnian</th>
<th>Esperanto</th>
<th>Icelandic</th>
<th>Latgalian</th>
<th>Norwegian Bokmål</th>
<th>Scots</th>
<th>Tajik</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acholi</td>
<td>Breton</td>
<td>Estonian</td>
<td>Iloko</td>
<td>Latvian</td>
<td>Norwegian Nynorsk</td>
<td>Serbian</td>
<td>Tamil</td>
</tr>
<tr>
<td>Afrikaans</td>
<td>Bulgarian</td>
<td>Finnish</td>
<td>Indonesian</td>
<td>Ligurian</td>
<td>Occitan</td>
<td>Sicilian</td>
<td>Telugu</td>
</tr>
<tr>
<td>Albanian</td>
<td>Burmese</td>
<td>French</td>
<td>Interlingua</td>
<td>Lithuanian</td>
<td>Odia</td>
<td>Silesian</td>
<td>Thai</td>
</tr>
<tr>
<td>Arabic</td>
<td>Catalan</td>
<td>Frisian</td>
<td>Irish</td>
<td>Luganda</td>
<td>Papi</td>
<td>Sinhala</td>
<td>Tibetan</td>
</tr>
<tr>
<td>Aragonese</td>
<td>Catalan (Valencian)</td>
<td>Friulian</td>
<td>Italian</td>
<td>Luxembourgish</td>
<td>Persian</td>
<td>Slovak</td>
<td>Triqui</td>
</tr>
<tr>
<td>Armenian</td>
<td>Central Kurdish</td>
<td>Fulah</td>
<td>Ixil</td>
<td>Macedonian</td>
<td>Polish</td>
<td>Slovenian</td>
<td>Turkish</td>
</tr>
<tr>
<td>Armenian Classic</td>
<td>Chinese (China)</td>
<td>Gaelic, Scottish</td>
<td>Japanese</td>
<td>Maithili</td>
<td>Portuguese (Brazil)</td>
<td>Songhay</td>
<td>Ukrainian</td>
</tr>
<tr>
<td>Armenian Eastern</td>
<td>Chinese (Taiwan)</td>
<td>Galician</td>
<td>Kabyle</td>
<td>Malay</td>
<td>Portuguese (Portugal)</td>
<td>Sorbian, Lower</td>
<td>Urdu</td>
</tr>
<tr>
<td>Arpitan</td>
<td>Chinyanja</td>
<td>Georgian</td>
<td>Kannada</td>
<td>Malayalam</td>
<td>Punjabi</td>
<td>Sorbian, Upper</td>
<td>Uzbek</td>
</tr>
<tr>
<td>Assamese</td>
<td>Crimean Tatar</td>
<td>German</td>
<td>Kaqchikel</td>
<td>Manx</td>
<td>Purépecha</td>
<td>Spanish (Argentina)</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>Asturian</td>
<td>Croatian</td>
<td>Greek</td>
<td>Kashmiri</td>
<td>Marathi</td>
<td>Quechua Chanka</td>
<td>Spanish (Chile)</td>
<td>Welsh</td>
</tr>
<tr>
<td>Azerbaijani</td>
<td>Czech</td>
<td>Guarani</td>
<td>Kazakh</td>
<td>Miahuatlán Zapotec</td>
<td>Romanian</td>
<td>Spanish (Mexico)</td>
<td>Wolof</td>
</tr>
<tr>
<td>Basque</td>
<td>Danish</td>
<td>Gujarati</td>
<td>Khmer</td>
<td>Mixteco Yucuhiti</td>
<td>Romansh</td>
<td>Spanish (Spain)</td>
<td>Xhosa</td>
</tr>
<tr>
<td>Belarusian</td>
<td>Dutch</td>
<td>Hebrew</td>
<td>Kichwa</td>
<td>Mixtepec Mixtec</td>
<td>Russian</td>
<td>Swedish</td>
<td></td>
</tr>
<tr>
<td>Bengali</td>
<td>English (Canada)</td>
<td>Hindi</td>
<td>Korean</td>
<td>Náhuat Pipil</td>
<td>Santali (Ol Chiki)</td>
<td>Sardinian</td>
<td>Tagalog</td>
</tr>
<tr>
<td>Bodo</td>
<td>English (Great Britain)</td>
<td>Hungarian</td>
<td>Lao</td>
<td>Nepali</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Project Fluent

● How we localize Firefox
● A localization system for natural-sounding translations
  ○ Developers can write in English
  ○ Translators can produce appropriate translations for more grammatically complicated languages :)

```ruby
# Simple things are simple.
hello-user = Hello, {UserName}!

# Complex things are possible.
shared-photos =
    {UserName} {photoCount =>
        [one] added a new photo
        *[other] added {photoCount} new photos
    } to {UserGender =>
        [male] his stream
        [female] her stream
        *[other] their stream
    },

UserName: Anne
UserGender: male
photoCount: 3

hello-user: Hello, Anne!
shared-photos: Anne added 3 new photos to their stream.
```
Fluent for developers

- Developers write define the properties in English in text files
- Can be localized declaratively through the DOM using attributes
- Or programmatically through a JavaScript API

```javascript
# Variables:
# $count (Number) - Number of tracking events blocked.
# $earliestDate (Number) - Unix timestamp in ms, representing a date. The
# earliest date recorded in the database.
graph-total-tracker-summary =
  { $count ->
    [one] <b>{ $count }</b> tracker blocked since { DATETIME($earliestDate, day: "numeric", month: "long", year: "numeric") }
    *[other] <b>{ $count }</b> trackers blocked since { DATETIME($earliestDate, day: "numeric", month: "long", year: "numeric") }
  }
```
Fluent for translators: Pontoon

Localize Mozilla

If you want to make Firefox available in your language, join a global community of localizers from all over the world, track progress of various Mozilla localization projects, and more; Pontoon is there for you.

Start Localizing Now or Take a Tour
Translating a string in Fluent

- UI Presents English source text
- Allows translator to edit target language text
  - Notice that there are more pluralizations in Lithuanian than in English

| one (e.g. 1) | Nuo { DATETIME($earliestDate, day: "numeric", month: "long", year: "numeric") } buvo užblokuotas |
| few (e.g. 2) | Nuo { DATETIME($earliestDate, day: "numeric", month: "long", year: "numeric") } buvo užblokuoti |
| other (e.g. 0) | Nuo { DATETIME($earliestDate, day: "numeric", month: "long", year: "numeric") } buvo užblokuota |

COMMENT Variables: $count (Number) - Number of tracking events blocked. $earliestDate (Number) - Unix timestamp in ms, representing a date. The earliest date recorded in the database.

REQUEST CONTEXT or REPORT ISSUE
Internationalization in Fluent

- Fluent needs to localize numbers and dates too!
- Built-in functions NUMBER, DATETIME to handle this
- Based on ECMA-402 APIs

**DATETIME**

Formats a date to a string in a given locale.

Example:

```javascript
today-is = Today is ${DATETIME(@date, month: "long", year: "numeric", day: "numeric")}
```

Parameters:

- hour12
- weekday
- era
- year
- month
- day
- hour
- minute
- second
- timeZoneName

Developer parameters:

- timeZone

See the Intl.DateTimeFormat for the description of the parameters.
How is this all Implemented?
ICU4C

- International Components for Unicode for C (ICU4C)
  - Also an ICU4J for Java
  - And an ICU4X that we’ll discuss later

- Enormous C (and C++) i18n library, first released in 1999

- Used everywhere!
  - Implement internationalization in SpiderMonkey and Gecko
  - And in Chromium, WebKit, etc.
  - And in OS X, Linux, Adobe products, etc. etc.
ICU4C

- Provides a lot of functionality
- But it’s monolithic
  - Very hard to remove code or data that you don’t use
  - Large dependency for browsers
- API mismatch with ECMA-402
  - Lots of work setting up calls, handling options and converting results
Unifying Firefox’s Internationalization Code

- At the beginning of 2021 Firefox effectively had three i18n implementations:
  - One in SpiderMonkey
  - One in Fluent
  - And one in Gecko

- ICU4C calls scattered throughout the code base
  - Lots of code duplication to handle parameters and return values

- No guarantee of consistent results between SpiderMonkey and Gecko
  - e.g. using different options when formatting numbers and dates in SpiderMonkey and in Fluent
Unifying Firefox’s Internationalization Code

- Moved all ICU calls to a single library usable from SpiderMonkey and Gecko
- Developed interfaces based on ECMA-402
  - Hide ICU4C complexity from developers
  - Get rid of duplicated code
  - Make sure we’re consistent when calling into ICU APIs
- Largely used existing SpiderMonkey code
  - SpiderMonkey code well tested, thanks to test262
Unification Results

- Got rid of a lot of duplicated code
- Had almost no regressions while doing so
- Full set of ECMA-402 APIs are now available to localize the browser
- And we can easily experiment with other internationalization libraries
Experimenting with ICU4X in Firefox
ICU4X Project

- Reimplementation of ICU4C designed around the needs of the web
  - Smaller, faster
  - Data is not monolithic
  - APIs based on ECMA-402
- Initial implementation in Rust, support for other languages through FFI
- Under development by Google and Mozilla since 2020
  - 1.0 Release planned for end of June 2022
Experimenting with ICU4X in Firefox

- In June 2021, ran experiments using ICU4X in SpiderMonkey
  - Intl.NumberFormat
  - Intl.DateTimeFormat
  - Intl.PluralRules
  - Intl.Locale
- Looked at performance, memory use and correctness
Intl.DateTimeFormat Performance

Profied Format and Calendar Operations
accumulated time (ms)

- Calendar Operations
- Format

![Bar Chart](chart.png)

- ICU4C: 229
- ICU4X: 46
- Total: 92
Intl.DateTimeFormat Memory Use

DateTimeFormat - Heap Memory Usage

Heap Memory MB

<table>
<thead>
<tr>
<th></th>
<th>ICU4C</th>
<th>ICU4X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory MB</td>
<td>100.9</td>
<td>18.95</td>
</tr>
</tbody>
</table>
Correctness

- Ran correctness tests using Test-262 for:
  - Intl.Locale
  - Intl.PluralRules
- Other implementations were not complete enough for results to be meaningful
- Handful of failures
  - Mostly around locale canonicalization
Some Unexpected Results

- Intl.NumberFormat memory use was worse with ICU4X!
  - 78.0 MB with ICU4X vs. 21.3 MB with ICU4C

- Smaller ICU4X objects meant less frequent Garbage Collection
  - ICU4X
  - ICU4C:

- This was a side-effect of the integration with SpiderMonkey
  - Need to estimate object size for Unicode objects in SpiderMonkey
  - But this estimate affects the behaviour of the garbage collector
  - Not an ICU4X issue :)
Overall Results

- In general performance and memory use are much better
- But need to be cautious
  - Incomplete implementations of DateTimeFormat and NumberFormat
  - Integration was a little hacky
  - It would be interesting to benchmark again after ICU4X 1.0
- ICU4X was promising enough to continue our development
  - Collator
  - DateTimeFormat
  - Segmenter
ICU4X Text Segmentation
Using ICU4X for Segmentation in Firefox

- ICU4C is not suitable for Firefox segmentation
- Layout engine does not use ICU4C for segmentation in the browser
  - Need to adjust line breaking results according to CSS properties
  - ICU4C data size is too large, because each word-break and line-break combination requires its own data set.
- Using ICU4C for Intl.Segmenter but not layout would mean inconsistent results
  - Exactly what we wanted to avoid with our unification project
Using ICU4X for Segmentation in Firefox

- Segmentation is also a great use case for experimenting with ICU4X
  - Not just a faster version of ICU4C
  - Does something we can’t currently do with ICU4C
- Will give us consistent segmentation:
  - Across platforms
    - Our Layout implementation is platform specific
  - And between Layout and SpiderMonkey
Implementing Segmentation in ICU4X

● Rule based
  ○ Algorithm determines whether a character is a break
  ○ UAX #14: Unicode Line Breaking Algorithm
  ○ UAX #29: Unicode Text Segmentation

● Dictionary based
  ○ Look up codepoint in a trie to determine where to break
  ○ Used for Asian languages

● Neural network models
  ○ LSTM models trained on dictionaries
  ○ Space/Time tradeoff: slower, but require less storage
Integrating the ICU4X Segmenter

- Will start after ICU4X 1.0 Release
- Already refactored existing Layout code to have an ECMA-402 like interface
- API surface is small
- But need to figure out how to handle data packaging in Firefox
  - Something we didn’t try in our earlier experiments
Experimenting with the ICU4X Segmenter

- Write microbenchmarks for word and line breaking
- Analyze impact on code and data size
- Enable for Linux Nightly builds
  - Behind a preference
  - Linux segmentation is currently the least functional, so the lowest risk for us
  - Check for regressions, performance problems, etc.
Implementing Intl.Segmenter

- Start once we’re happy with the ICU4X integration
- Should be straightforward
  - Have a work-in-progress implementation using ICU4C for an older version of the spec
- Validate against Test-262
Conclusions

- Internationalization is important for the browser and in JavaScript
- By unifying our implementation, we’ve simplified our code base significantly
- We also set ourselves up to try out new implementations
- Because of this work, we can easily try ICU4X for text segmentation
Questions?