ICU4X Segmenter Investigation

Authors: Ting-Yu Lin
Last Modified: 2021-04-16
Status: Draft

Objective

Text boundary analysis is the process of locating linguistic boundaries while formatting and handling text. ICU4X would like to provide a unified segmenter (similar to ICU's BreakIterator) for client-side software for the following four types of boundary.

1. Character Boundary
2. Word Boundary
3. Line-break Boundary
4. Sentence Boundary

The unified segmenter can power the following possible usages in Firefox:

- Layout (line/word breaking with CSS properties line-break and word-break considered)
- Forms/Editor (double-click to select a word, text editing, caret positioning)
- Javascript (SpiderMonkey) API (Intl.Segmenter)
- Firefox UI (find in page, look up dictionary on macOS -- usually via three-finger tapping on a word)

Requirement

- The segmenter interface needs to accept a locale and a break type (character, word, line, sentence), and return a Rust iterator.
- Given a string (UTF-8/UTF-16/Latin1), the segmenter should iterate the possible breakpoints for a given break type.
    - Firefox uses UTF-16 and Latin1 internally.
    - icu4x has string_representation.md on this topic.
    - For example: uax14_rs is a UAX#14 implementation supporting CSS properties, written by a Mozillian Makoto Kato. It supports UTF-16, Latin1, and Rust char (UTF-8) BreakIterators.

      It uses a python script to generate UAX#14 rules table to drive the finite state machine. The runtime customization for CSS properties is handled before matching the pre-generated rule tables. The iterator `next()` method returns a position within the string (as `usize`) as the next possible breakpoint. See this test for the usage of the line breaker.
- The segmenter should find the next break opportunity efficiently, ideally in constant time. That is, the number of rules to match shouldn't be a factor of the running time.

Approach

There are three general strategies of boundary analysis for different languages:

- Rule-Based algorithms based on Unicode UAX#14 and UAX#29
    - CLDR provides additional per-locale data for additional adjustments
- For South-East Asian (***SEA***) languages like Thai, Burmese, Lao, and Khmer, traditionally they need dictionaries for word breaking. Recently, machine-Learning based approach is developed for Thai and Burmese in lstm_word_segmentation, and ICU4X has an experimental crate for it.
- For Chinese, Japanese, and Korean (***CJK***), they still need dictionaries for word breaking until a machine-learning based approach is proven possible.

To implement a rule-based line segmenter (or other types of segmenter), we need the following two sets of data/function:

1. A function to map a certain codepoint to its line break property value, e.g. **f(codepoint, line_break_property) -> line_break_property_value**
2. Line break rules to build the segmenter.

How to get **f(codepoint, line_break_property) -> line_break_property_value**?

UCPTrie (Code Point Tries) is the right data structure to use. icu4x#132 tracks the work.

Runtime tailoring/customization support of ICU4X's segmenters

This section discusses the scope of runtime customization ICU4X's segmeter would like to support. Here we use line break iterator as an example because it's the most complex one. Character/word/sentence breaker may need similar customization.

One application of the line break iterator is implementing line breaking in the browser's layout engine. ICU4C already considers "line-break:normal", "line-break:loose", and CJ/non-CJ in current break rules. However, to implement a CSS compliant line breaker, we need more than that. **Here is a supplemental document on why runtime customization is essential to ICU4X's line breaker for web engines.**

Question: Is runtime customization in the scope?

Option 1: No, we don't want to support runtime customization, and want to implement it on top of ICU.

If supporting runtime customization for the line breaker is out of scope, it is convenient to reuse ICU4C's break rules to implement feature-compatible segmenters in ICU4X.

A possible approach is to write a tool in ICU4C to generate state tables suitable for Rust to consume. It's not recommended to implement a full RBBI rule builder unless we have to.

Pros:

- Only need one runtime for all types of break iterators because the state transition table's format is the same.
- Easy to update against spec change if ICU4X follows closely with ICU4C. That is, the burden to update against the latest spec is on the ICU4C side.

Cons:

- Offer no customizability at runtime, which is hard to use for web engines that need to switch line breaking behavior according to CSS properties.
- Unless we implement a full RBBI rule builder, the user cannot customize via rule data like ICU4C.

Side note: Q&A and comments on ICU's break rules and segmenter model.

- How easy is it to tailor the ICU break behavior?
- Any known issues in ICU for customization that are hard to implement because of the current design?
- Is it easy for ICU to update rules against spec updates?
  (The answer explains the difference between ICU rules and UAX rules, and the pain points to handle ZWJ.)
  (See also updating ICU's built-in Break Iterator rules)
- Concerns on building the generic rule based break iterators.

Option 2: Yes, we do want to support runtime customization and implement it from scratch.

Support runtime customization directly in the code, and provide flags to switch the behavior.

Example 1: Override non-tailorable rules in UAX#14
line-break:anywhere. Quote from the CSS Text spec, "There is a soft wrap opportunity around every typographic character unit, …., disregarding any prohibition against line breaks, even those introduced by characters with the GL, WJ, or ZWJ line breaking classes". (Note: GL, WJ, and ZWJ are all non-tailorable per UAX#14.)

Example 2: Transform a line breaking class to another
word-break:break-all. Quote from the CSS Text spec, "Breaking is allowed within "words", …, any typographic character units resolving to the NU ("numeric"), AL ("alphabetic"), or SA ("Southeast Asian") line breaking classes [UAX14]) are instead treated as ID ("ideographic characters") for the purpose of line-breaking." (Convert NU, AL, and SA to ID)

Pros:

- Easier for browsers to implement line breaking. Here a code snippet in uax14_rs crate implementing the word-break:break-all.

Cons:

- It may be a bit strange to support various CSS properties directly in the segmenter's code, but we can invent whatever flags with proper names to switch line break's runtime behavior.

Conclusion: TBD

(Ting-Yu: I love to see we go for Option 2)

On generating break rules that conform to UAX14 and UAX29 specs.

Option 1: Use line.txt as the base rule set for line break iterator. Similar for other breakers.

Pros:

- Assume ICU4C's break rules conform to the spec. We don't need to read the spec in order to implement the break iterators.

Cons:

- Require the knowledge of ICU4C's internal to write a tool to dump data for ICU4X to consume. This can be hard to debug because there is no obvious connection between the dumped data and the spec.

Option 2: Generate the rules from scratch by reading the spec.

Reference implementations:

- (UAX14) uax14_rs uses a python script to generate line break rules into a rule table. The runtime customization is implemented directly in the finite state machine, such as transforming one line breaking class to another. They are processed before matching the pre-generated binary rule table.
- (UAX29) unicode-segmentation implements the segmenters manually by encoding the rules in the finite state machine. See word.rs for its word breaker.

Pros:

- Without depending on ICU4C like option 1. We can have a chance to provide feedback to the spec if we value an independent implementation.
- Easier for external contributors to contribute to the implementation because they can read the code and compare it with the spec.

Cons:

- Need someone to get familiar with the spec to implement. Each breaker (character, word, sentence, line) needs to be implemented separately.
- Can be tricky to implement CLDR's tailorings because the tailoring rules are based on ICU's break rule syntax. An example of CLDR tailoring of Japanese.

Conclusion: TBD

(Ting-Yu: I prefer Option 2.)

Other questions

- The name of the API? **Segmenter**, BreakIterator, or something else?
    - Conclusion: Segmenter
    - The integration branch uses **icu_segmenter** as the crate name, and `LineBreakIterator`, `LineBreakIteratorUTF16`, and `LineBreakIteratorLatin1` as the iterator names.
- If we decided to generate break rules, should we plug those rule tables into the segmenter via ICU4X's DataProvider architecture?
    - Conclusion: We should use DataProvider.
- Other necessary todo items before merging the integration branch into icu4x?
    - https://github.com/aethanyc/icu4x/issues/1

Reviewed By

-

Document History

- 2021-04-16: Revised the document based on feedback from commenters.
- 2021-04-02: Added section "On generating break rules that conforms to UAX14 and UAX29 specs.".
- 2021-03-31: Added section "Tailoring / customization support of ICU4X's segmenters"
- 2021-02-25: Mentioned the integration branch vendoring uax14_rs. Revised based on comments.
- 2021-02-21: Initial draft

2021-04-21/22 Notes (Deep Dive Part 2)

Agenda:
- Decide the scope of runtime customization of ICU4X's segmenter.
- Decide the approach to generate break rules.

Discussion:
- Link to document: Runtime customization is essential to ICu4X's line breaker for web engines
- Frank: Why are the switches for customizing segmentation needed at runtime? Why can't it be done at build time?
- Zibi: Problem exists where you have a web page where different paragraphs have different "word-break" property values (loose, normal, etc.). Then you want to be able to switch between various segmenter instances where a majority of the rules are shared

- Manish: For many of the word-break CSS property values, it’s not as much about switching to a different tailoring of an algorithm, it’s more about switching among grapheme iterator, word break iterator, line break iterator, which is easier to manage
- Markus: It sounds like ICU supports what is required here. and Andy and Frank have worked to make these iterator rule sizes relatively small (30-50 kB). It would be nice to share some of that data, although it’s tricky to build a FSM. The benefit is speed, the FSM is fast to execute. If we are running something inside a browser, it is probably even more sensitive to line- and word-breaking runtime performance. I am concerned if we are de-emphasizing the runtime aspect
- Mark: Agree with Markus. We need to be careful in #2 (in the list at top of “Runtime customization…” doc). For using “word-break: break-all” vs. “word-break: break-word”, you apply a different iterator (break-word might break in the word so needs a grapheme break iterator). So that takes a little more examination. As far as size goes, work has gone into bringing the size down. When iterating over text, first you map code points to unicode properties (?), and then map properties to categories of characters. But you can then change the mapping of properties to characters as needed without changing the code, ex: for Indic scripts, the Aksara property
- Frank: From what I can see, I urge you to reconsider the concept. When I hear "runtime customization", it sounds to me like if you have a large set that the customer wants to customize. Like, if CSS lets you add feature overrides. But what you put in here is rather large, but not huge, like 8-10 types of word break rules. It's not 100 or 1000 combinations. So the issue is, if you think about runtime customization, and you want to do this with runtime, ICU4C can do this today. You just ship the precompiled rules. In my view, runtime customization should not be a requirement for this use case. That should be build time. What you really want is build time customization, and data that can be shared. Not actually runtime customization.
- Zibi: You almost said exactly what I wanted to say. I want to rephrase. It seems like you are positioning a tradeoff between performance and memory. You emphasize performance and claim that memory cost is not as drastic as we are worried about. Is that correct?
- Markus: Disk memory, not heap memory. The state table is a large set of arrays that can use zero-copy deserialization.
- Zibi: In this case, I’d like to ask Makoto, what made you not follow ICU4C for the break iteration library?
- Makoto: Our engine also uses a state machine. The state machine is as compact (?) as ICU is now. But the data is not important for heap memory (?). We create a runtime thing that combines rules to support 20 rule sets. It makes a large table now. So I think a small state machine set, such as a processing rule and a main rule and a post-processing rule could be more compact. I'm not sure.
- Zibi: Does it mean that you would be open to explore what Frank suggested? Is the ability to share immutable portions of the rulesets at runtime a good suggestion?
- Makoto: Yeah… although, I am not writing it… I would like to consider a little bit of ??. I don't have an answer now.
- Frank: Are you still using the wrbrk stuff?
- Zibi: Makoto began writing a Rust crate, uax14_rs, which was an attempt to advance wrbrk for UAX 14. And that was intended as a replacement for that. So the discussion was whether we can make that part of ICU4X. It sounds like uax14_rs has its own way for overrides, but Markus and Mark are making claims that runtime

performance is more important to consider. Also, I understand that Blink is overriding ICU4C.

- Ting-Yu: I understand that it's convenient for ICU4C to implement a small runtime. To implement different CSS properties the ruleset is really hard to understand, like 300 rules in line break iterator. In ICU there is one runtime for that and no place to insert customization code in the loop or FSMs. In Servo each break iterator is written by hand. It either encodes a bit table like UAX14, or returns in the loop.
- Mark: I think we might be talking past one another. There would be certain extensions needed in ICU to make customization of properties easy, but that wouldn't be too difficult to do. It's all rule drive (data driven), the question is how many customizations are needed? The model described in item #2 of the first list in the other doc doesn't seem to reflect reality. You don't need 20 different instances of break iterator, that's not required.
- Rich: One thing that I think plays into this is, in the question of run-time customization, is how many and what kind of variations are you expecting or needing, and how often do you expect that those algorithms change over time. If they're relatively stable over time, I don't think that runtime customization is a high priority. If the variation can be expressed by changing character categories like Mark is saying that greatly reduces (?). Where are those requirements coming from?
- Ting-Yu: There are a lot of requirements in the CSS spec that require changing break rules depending on the property value. I don't know if they're stable enough; the CSS spec is frequently changed in the CSS WG. It's easy to transform one break property to another in code. You can write some code that does the transformation before the rules fall back to the default rule set.
- Rich: What I'm still not getting is a sense of what kind of customization is required. I think it would be good to list out a set of actual examples. What, in practice, have you run into that is motivating these requirements?
- Ting-Yu: Here in this doc, I have two examples (in the section "Runtime tailoring/customization" -> Option 2) from the CSS spec. Example 2 is such an example ("transform a line breaking class to another").
- Mark: But that's just a different iterator.
- Zibi: I wanted to verify with the group's-- Ting-Yu do I understand correctly that CSS sentences are like this. We need to load build-time rules into memory, but Mark is saying we can solve it on the code level.
- Mark: Two things. (1) If you are really allowing breaks in between any two grapheme clusters or any two code points, that's something that's done by a different iterator. It's not done by a line break iterator, but it's available. (2) I agree with Zibi that it's about using different iterators at different points. I can check in code whether or not I'm going to break that word apart if I see something in the middle.
- Frank: Think about it this way, the key thing I want to communicate is that you have to have 5 different specs. If you implement 5 different iterators, assuming our implementation will make it easy to share data, that I think is a good requirement. But deal with it in the build time. Then what you can do is write five different tests to ensure that it meets the requirements. If your goal is to support a run-time customization requirement, the bar is much higher. You need to make sure you have an engine that is so flexible to make sure that it performs correctly in all of the different ways. And then you need to test it in five different runtimes. The bar to design something with runtime customization is a much more difficult task than if you

address this in the build time. The required engineering and quality bar is much higher, and I think it's overkill. I think what you build in ICU4X could be shared, and that would be a good goal, but it should be a build-time goal, not a run-time goal.

- Zibi: I have to say without in-depth knowledge on the domain, your phrasing is appealing to my mind. I would love to see it documented somehow, like posting it in the Github issues in ICU4X. It sounds like you want the system to be lightweight, with a set of base rules and a light set of overrides that can be applied.
- Zibi: In this meeting, we have Ting-Yu and Makoto who are exploring the system for line breaking that has been written in a certain way, and Mark and Markus and Frank not seeing the same set of priorities. So do the 2 groups not see the reasons for why the other has implemented things in the way they have?
- Frank: Let me say something different than Zibi asked. I wrote the original lwbrk at Mozilla probably 22 years ago. The difficulty at that time was the adoption later on. I did once consider whether we could switch to ICU. It's more about the API model. The HTML coming in is in different frames and you have to decide the line break between them. So the memory model at that time in Mozilla was you have two API. One is you have a run of text, and walk forward. That's like ICU. The other condition is you have two data frames, and you want to see if you can break in between. Another condition is that if you have English text, like "Apple". But the first "A" is inside a tag, and "pple" is somewhere else. So you need to be able to pass in "A" and "pple" and check if you can break there. I'm not sure how to fit that into ICU BreakIterator. It becomes awkward to decide how to switch over. When I was working on lwbrk, I was following the line break code at the time in Mozilla. So if you guys are thinking about how to replace it, that's something to consider. I'm pretty sure the ICU4C API is not optimized for that kind of model. In particular, when the memory is fragmented, and you have to jump around.
- Markus: I want to point out that ICU implements CLDR rules not raw UAX 14/29 rules. The UTC defers to CLDR for tailorings and for trying out proposed changes. ICU does have rules for the CSS variants. ICU4C has since added support for "UText". you could make one of those in such a way that it omits the markup. In ICU4X you could implement the runtime code right away such that it iterates over the plain text chunks.
- Rich: I want to go back and talk about history a little bit. One of the comments that I heard early on today is that ICU's rule language for describing BreakIterator behavior is complicated and difficult to maintain. I wrote the first rule-based break iterator (RBBI), and I am the first to say that it is awful, I would like to see something better. At the same time, the whole concept of taking a human-readable set of rules and converting it into a state table is always going to be expensive, and so it's important to steer away from depending on that as the cornerstone of how your break iteration works. Customizing break iterators so that they can share configuration sounds good, but I think that you can achieve that through sharing the rule sets themselves. I think what Mark said about the pathological case where breaking happens in the middle of a word is also useful.
- Manish: I've looked at the UAX rules in CLDR and when I implemented unicode-segmentation (Rust library) support for the more recent emoji stuff, I did a lot of work in optimization. Those rules are in a certain, like, there's a whole world of difference between code that can apply those rules and code that can do it efficiently,

especially for flags and stuff like that. So I think it will be a lot slower to support arbitrary tailorings. But I think we need to. So maybe we can do a hybrid approach.

- Manish: The other thing is that there are ways to work with ropes and streaming APIs and such. Raph Levien at Google uses ropes to do that efficiently. There are ways around the " ‘Apple’ where ‘A’ is bold" problem.
- Ting-Yu: We want a stable rule set. If CSS spec is stable enough then we can do customization in code, and use a flag to customize behavior during runtime.
- Mark: To add arbitrary customizations by changing properties is easy. Adding arbitrary customization to change rules is a pain. In theory, we could build a rule parser that would be very flexible and take a simpler format like what's in the specs. But that's a big project. Not scoped out. Not necessary to do at this time. So I think if we scope this to, provide a mechanism to change the properties, that will give us lots of flexibility to get speed, size, and some degree of customization.
- Rich: I agree 100%
- Ting-Yu: Ideally, we can change human readable rules at runtime. Is that what you're saying?
- Mark: That's not what I'm saying, but I have to leave unfortunately.
- Zibi: Manish, do you believe that the issues / problems that we’re raising, can it be implemented in the ICU4C model?
- Manish: Yes, I believe so, except for the part about the performance of customizations, for which I’m not sure. I think it’s doable, that middle ground to support tailoring.
- Zibi: Would you be able to help?
- Manish: Sure? But I do not want to be considered an authority on the matter.
- Frank: I just have a clarifying question for Mark who is away, so I’ll ask Rich. Does adding customization for properties meaning, I can somehow change a character to belong to another property?
- Rich: Right. You can change the property table easier than the state table.
- Frank: So if there are 5 punctuation marks in one class that now belong to a new class. Is that right?
- Rich: Yes
- Frank: So it seems that what Mark suggests is that if you want to have some kind of customization, then it should be narrowed down to the customized character property mapping, and somehow inject the custom property mapping you need. You should see whether that fits. What I’m seeing right now in ICU4C is that we have 4 or 6 different rule files for the line breaks because of this, there are some small differences (ex: 3 character differences). And maybe those issues could be solvable by this proposed approach.
- Andy: Just expanding about what’s easy to do at runtime in ICU. The mapping from a character category to what ICU uses gets munged at the builder time. It would be relatively easy at runtime to make one character behave as another character. But to make one character have another character class could actually be tricky. It’s a little more restrictive than what Mark may have been implying, but it could still work.
- Responding to Rich’s comments on the RBBI rule syntax, it’s very difficult to maintain, and it comes from all the rules being applied at once, which is the magic of the FSM. But the UAX 14 spec actually describes applying the rules in order. But I don’t know whether implementing the spec would have a big impact on performance, but it would simplify a lot of the complexities in ICU RBBI.

- Shane: I want to get a bit more clarity on the question in terms of CSS: Ting-Yu found some code in Blink where it was doing some kinds of overrides on ICU, and I wanted to get more clarity on what those overrides are for and what scope they have.
- Ting-Yu: There is no way to switch between line breaking class to another in the current ICU architecture, so they wrap the ICU line breaking functionality internally. So when there is a need to break in the middle of the word, they do so, but they fall back to using ICU break iterator when necessary. So they wrap the ICU break iterator.
- So when Rich mentioned inventing a different language for the rules, that might help because ICU4X doesn't want to rely on the ICU rule syntax.
- Shane: As someone who doesn't know much about segmentation, here is my impression: It sounds to me like Markus, Frank, Rich, Andy et al. have been saying that toggling between the five different break styles can and should be done in code. Other customization like unicode properties is fairly easy to plug into the existing ICU algorithm. There is still low-hanging fruit to reduce the data size of the ICU state tables. Ting-Yu is saying that rather than having to write that part of the algorithm in code, we should be able to express that in a rule language. That rule language could be similar to the spec, or something that we write custom and externally. In order to support that rule language, it's hard to pre-build the rules at compile time, and we need to do it at runtime. Is that an appropriate summary from an outsider in this subject area?
- Ting-Yu: To implement these CSS properties, I would advocate to do that in code. We don't need to invent a whole new language to express those requirements in data. Imagine a small contributor who wants to break a line in a different way, they can submit a PR and make the line break iterator be what they want it to be.
- Frank: Where is the Blink code that you are referring to? I can investigate.
- Zibi: It's linked from the document "Runtime customization is essential…" -> Link
- Ting-Yu: I think we may have a different concept on the runtime tailoring. I want to address some of the issues with ICU4C. Back in our previous deep dive, there was the suggestion to dump the rule data from ICU4C. But I could also implement the state machine from scratch. Because then I can add the customization in code. I want to get an answer from these two options for ICU4X.
- Rich: I don't necessarily disagree with what you're trying to do, the way you have it described in this document. But I would suggest again 2 things: (1) don't underestimate how big a lift it is if you are going to design everything from scratch. That is a _big_ job that will require a boatload of testing. (2) A setup based on building your data structures on the fly at runtime strikes me as likely to have performance problems. And I think if you go this route, performance is going to be an issue you have to focus on hard. I think it is a worthy goal, but I think it is unnecessary to get something going quickly.
- Markus: building data at runtime also incurs a higher heap memory cost
- Markus: My thoughts are that there are a lot of people who have worked on segmentation more than I have, and I defer to them.
- Zibi: can we look back at the doc "Runtime customization is essential…" ? It seems to me that the Unicode reps here claim that #1 and #2 from this doc say that those could be true, but it may be more work than the benefit. For the second one, Mark is claiming that is not 4x5 rule sets, and Frank is saying it's possible we can reduce in memory the amount of data we store. It leaves us with #3, it's not just that Mozilla is

not using ICU4C directly, but Blink is also doing something custom. It would be nice to end up with segmentation that doesn't force browsers to do custom work. It sounds like the ICU model can support that, and I want to see if the Unicode experts agree that what Blink is doing is unnecessary or fixable.

- Frank: I just want to clarify that what I say is that goal isn't runtime optimization, the goal is to make it small. It may be difficult to do on the compiled table. Each character has a particular property. As Andy says, you can perhaps make it so that a character inherits the info of another character, but Mark was saying something else. I have a hard time knowing how to fix it because of the reasons that Andy stated.
- Andy: The ICU rules are extremely difficult to modify because of the parallel application of them. If a sequential algorithm following the UAX algorithm were possible to write efficiently, that would be great, but it's difficult to do that. So if performance is not as much of a concern, then runtime customization could be possible. But otherwise, the ICU tables are hard to customize.
- Rich: I wonder if it's possible to convert algorithmically from a set of rules that apply sequentially to a set of rules that apply in parallel.
- Andy: I fought that problem so hard, I think it would be a major technical breakthrough if that can be achieved.
- Frank: When we talk about performance, there are 2 kinds. (1) how to compile the rules. ICU does that at build time. So the performance for that is not a huge impact to the user, although people can still do that. (2) runtime performance. Both types of performance are important. You might think you only care about runtime performance, but if you build the rules at startup time of an app, we actually get into trouble in V8 because of startup compilation time.
- Zibi: I can say 2 things. What I haven't heard anyone from experience with ICU4C recognizing any of the points in those documents as valid concerns that should be addressed. What I hear is that they believe that this can be solved using ICU4C. But I don't hear Ting-Yu agreeing this is true. So my concern is that we leave this meeting holding our same positions instead of converging. I was wondering if someone else can help bridge that gap, but I don't hear that either, so I don't know what the next steps are.
- Ting-Yu: The issue is that the break iteration in Mozilla is written from scratch, but reading ICU4C rules is difficult. Do we want to build up a binary set of rules using UAX 14 and 29, or do we want to read the ICU4C binary files directly?
- Shane: I think the ICU4C people on this call are aligned that the issues that Ting-Yu are laying out are noble goals. But I don't think they see implementing them as described will be large and not an efficient use of manpower. Ting-Yu, do you agree that they are overestimating the work involved?
- Ting-Yu: I feel that it may be easy because Makoto demonstrated in his crate that it is possible to do. (?)
- Frank: there is one thing I hear Ting-Yu, which is that the ICU Break Iterator rules are hard to implement. But some complexity comes from the spec, doesn't matter how you implement it. The other question is whether a new implementation can make some things easier, you don't bind yourself to the rule string syntax. Another thing is what Andy said, UAX 14/29 defines the rules in order, but ICU's execution tries to parallelize that state machine. Instead of trying to design this way, try to prototype, and measure it, you see the performance and see whether you are happy with it. But I do think Andy is right, once you try to apply those rules in sequence, it will slow

down a lot. Applying it in sequence becomes 10x easier to understand, but the cost is in runtime performance. So try it, write your tests first, then prototype, benchmark it, see what you get. At least do so on some small scale, then you learn more. This is my suggestion.

- Markus: the parser is complex, but exists. If offline building is sufficient, then using the output of the icu code is easy and attractive.
- Shane: I had a question maybe for Makoto. I understand lwbrk, which is the break engine in Firefox. Does it implement the runtime tailoring the way that Ting-Yu is describing here? If so, can we benchmark lwbrk? Have we run those tests before?
- Rich: I want to second Frank's comments wholeheartedly. I am just here to make suggestions and would love to be proven wrong on some of the things I've said.
- Daniel: I also agree with what Frank and Rich have said. The way ahead is through prototyping.
- Shane: On the point about prototyping, I don't have a great sense for how far away a prototype is that we can actually compare and benchmark. I feel like we have a huge amount of expertise that we can build on. If we allow Ting-Yu to repeat the same trial and error that has already been done, I don't know what it would look like in terms of timelines. On the other hand, we could take what people have said on faith that they might not work, or we could let Ting-Yu validate them, but do we have a sense on how long that validation is going to take?
- Makoto: I don't have a benchmark, so I think I need to benchmark to know, so I agree with the others.
- Dan: Could Makoto's work serve as a prototype for quick benchmarking?
- Makoto: When I was prototyping uax14_rs, I wrote a benchmark. Using Latin1 and UTF-16 (?) Thai language. Gecko code uses platform code for languages like Thai, Lao, etc. I'll try it [benchmarking a prototype].
- Zibi: I want to get a conclusion and next steps and call to action, and then turn it into a time-bound commitment. Can we time-bound the prototype? People with expertise in the domain say it's the right idea with a huge amount of effort required. Is there something that we can do within a week, 2 weeks, that gets us to a better answer of whether we can get something into production within 6 months? That's a question for Ting-Yu and Makoto.
- Makoto: There are 2 problems with the code. Line breaking (UAX1 4) and word breaking (part of UAX 29). If you ship intl segmenter in Gecko, then you lose things like dictionary segmenter and CJK segmenter (?). What is the priority for word break iteration and line break iterator?
- Zibi: Ting-Yu, were you prioritizing UAX 14, or UAX 29, more?
- Ting-Yu: Let's go with UAX 29, for prototyping, since line breaking is more complex.
- Zibi: Okay, I hear let's prototype using UAX 29, how long would that take to prototype?
- Ting-Yu: I'm not sure.
- Andy: The hazard with going with UAX 29 first is that it's so simple that you can get something working for UAX 29 that doesn't work for UAX 14. I had several attempts of getting approaches that work for UAX 29 to work for UAX 14.
- Frank: I think the most critical piece for runtime performance in browsers is line breaking.
- Rich: Yeah, you should do line break first.

- Frank: So if you want to prototype something to see if its runtime perf is acceptable, then you should start with line segmentation first.
- Shane: And that is the uax14_rs library that Makoto has been prototyping. So what more do we need to do to compare with ICU?
- Makoto: uax14_rs is currently easy to merge with Gecko, but before merging with Gecko, I will do comparisons when writing a prototype. I can't estimate a timeline, but I will be working on it next month.
- Shane: Does uax14_rs cover all of the use cases that Ting-Yu has laid out in his document here?
- Ting-Yu: I believe uax14_rs has covered all of the CSS properties, for word-break and line-break.
- Makoto: Yes, I have gotten the code to pass all of our web platform tests.
- Shane: Another action that would be useful is if one of the ICU4C Break Iterator experts could look at the uax14_rs code to see if it works and whether it is missing anything. Because it seems like we are talking past each other, and maybe there is an innovation there in Makoto's code that we haven't considered before.
- Manish: Maybe we can pair in such a way that I can explain what is going on in the Rust code, not trying to teach you.
- Shane: Is there anyone on the call who knows both the ICU4C BreakIterator codebase well and the uax14_rs codebase well?
- Frank: I think it's too early to tell, I think we can know better after looking at the results whether it will work, rather than trying to do too much upfront design. It's like a skunkworks, go ahead and try it, and the results will tell you more definitively what we're dealing with. I think that will be more productive. I think they understand everything we have talked about.
- Shane; I like where is standing. Thanks to Makoto's work on uax14_rs, we're most of the way there for a working prototype. Maybe the action for Ting-Yu is to benchmark that crate and make sure that it does everything that we want it to do. Choose use cases for algorithms, and then we can come back and compare.
- Frank: At the end of the day, you have to do prototyping, because people will always have questions about performance. Eventually, when you have a performance issue, you have to refer to a microbenchmark in a controlled environment. I agree with what Andy and Rich have said, the tricky part is not implementing it correctly, it's making it small enough on disk / memory and fast enough during runtime.
- Zibi: I am wondering how to balance long-term maintenance, technical excellence, and business needs changing. It might take longer to implement, but we need to have a recommendation for the future of lwbrk in Mozilla soon. But I don't know what the state is currently for that.
- Shane: The only path I see is what Frank said, we have discussed design but we haven't seen data, so we have to get data. The other approach I see is to either go just one side or the other (Makoto's uax14_rs is the way to go, or the ICU approach is the way to go), but neither would make the other side satisfied. I would like the bonsai approach, where we prune non-beneficial options, but I don't think we have enough data to do the pruning.
- Shane: You should write an app, and have it use uax14_rs and also have it use ICU, and compare how it performs.
- Zibi: I can help with creating such small apps to do such comparisons. What's a good way to test? Do we have a Wikipedia article?

- Frank: I think Andy has test data, but I don't know how representative is.
- Shane: I think just having one corpus is not enough. I think the perspective that Makoto and Ting-Yu are bringing is when these CSS properties change, so having data exercising that use case would be good.
- Frank: At some point, the implementer has to be happy that the performance is good enough. It takes a while to even reach that stage. Try to get a
- Zibi: Do you know what next steps are?
- Ting-Yu: I can work with Makoto on that.
- Zibi: Do you feel satisfied with what you have heard?
- Ting-Yu: Yes, after the previous deep dive, I thought that only the ICU RBBI was reasonable, but I'm happy to hear that we can try uax14_rs and compare.
- Zibi: Try to time bound it. Ex: in 2 weeks, we will have a sense of what path to go.

2021-02-26 Notes (Deep Dive Part 1)

- Subject: segmentation / breaking on {character, word, sentence, line} boundaries, defined in UAX 14 + UAX 29
    - Unicode Technical Committee (UTC) defers to CLDR for language tailorings and improvements before they make it back into the base standards. So don't just refer to technical reports / standards (ex: UAX 14, UAX 29), but also refer to incremental changes in CLDR in between official reports (standards / annexes) changes
- Main strategies of segmentation
    - Rule-based (RBBI - Rule-based BreakIterator)
    - Dictionary-based (ex: for SE Asian languages)
    - ML based (ex: ML model for Thai and Burmese segmentation)
    - Combination of rule-based + dictionary-based (ex: CJK languages)
- To implement UAX 14 (line breaking/segmentation)
    - Need line break rules
    - Need a function f(code point, line_break_property) -> line_break_prop_value
    - If you use the ICU4C RBBI builder code's generated & serialized data structures -- code point point + state table -- (write code to load/interpret the data structure), then you have a finite state machine that is constructed according to the Line_Break property baked in. (That is because the Line_Break property is an input to the builder for the RBBI). So you wouldn't need an extra work to read the Line_Break property if you go can reuse the ICU4C data structure for implementing rule-based segmentation
        - Technically, ICU4C RBBI builder code outputs a 3rd thing -- a copy of the original rules data -- in addition to the code point trie and state table, but we probably wouldn't have much use for that copy of the input rules
- Are there tests for RBBI in ICU4C/J?
    - Most of the tests are in data, and the data are shared between ICU4C and ICU4J
- Are there any special tests that we know of?
    - Andy has one implementation that is faster and optimized

    - Andy has another implementation that more directly pulls from Unicode technical reports' specifications that is slower, but is more believably accurate, and be can be used to validate the first implementation
- Note: Makoto has already integrated the ML segmentation approach for Thai and Burmese that Sahand worked on into the branch that Ting-Yu is working on (?)
- What is the extent of the unicode-segmentation Rust crate? How does that relate to RBBI?
    - How much functionality does it cover?
    - What is the data size?
- Way line-breaking is handled in diff projects
    - ICU4C - creates the RBBI, which is rule-based
    - uax14_rs - creates a table of line-breaking rules (~ a rules engine)
    - unicode-segmentation - hard-codes the rules, manually updated based on changes to Unicode specs
        - Goal is to be a self-contained UAX 29 implementation (character, word, sentence)
        - Makes things efficient like Emoji and regional indicators (?)
        - Some things won't be efficient, ex: like if someone added a virama rule
        - Most kinds of segmentation rules should be implemented and efficient
- Often-neglected segmentation rules
    - Ex: Rules that know that letter-dot-space isn't always a sentence break ("Dr. John")
    - Unicode technical reports (UAX 14 / UAX 29) allows for people to overlay/supplement the rules with these types of exceptions, but Unicode doesn't actively maintain the data of exceptions
    - So we shouldn't forget this, and we should consider how to support this
- Different tailorings we're talking about
    - for line-break, CLDR has data for
        - Japanese, some characters require exceptions
        - for CSS, support for some rules like 'loose' and 'strict'
    - for CJK language, we have a combination approach for line breaking (rule-based + dictionary-based)
    - also, ensuring that we prevent errors (ex: for sentence breakings, be smart about periods used as punctuation, or brackets not ending a sentence)
        - In ICU, to support locale-specific intelligence to avoid line breaks around periods in abbreviations, there is a separate mechanism called FilteredBreakIterator that is on top of BreakIterator that pulls in locale data before doing segmentation
    - at runtime, need to enable / disable depending on the use
        - Ex: different line breaking rules when you have a big body of text (want to fit text into a rectangular boxes, so okay to break in middle of words) vs. when you have the title of something (don't want to break in the middle of a word)
            - Comment: "i suspect that that's what the three CSS styles do. in icu, they are implemented as somewhat different rule files, each of which builds a different FSM"
              Response: "probably. i would need to look"

        - Mozilla has custom CJK break information that they would like to upstream. ICU BreakIterator didn't work very well for Japanese on the web, so Mozilla made a more customized segmenter.
    - ICU rules data for BreakIterator: https://github.com/unicode-org/icu/tree/master/icu4c/source/data/brkitr/rules
        - There really aren't a lot of language-specific tailoring. People don't really want different rules for different languages
        - These rules haven't changed much over a long time
- In ICU4C/J's generated data for RBBI, what is the data size for these extra
    - Attempted: Combined set of rules that create a superset of break opportunities, and create implementations as breaks on subsets of break opportunities.
        - Ran out of time to continue further investigation
    - Frank might be better to answer the question of data size. Guess: around 30-40 KB, after doing some optimization last year to reduce from the previous size by 30%
      (see detailed discussion below)
- Supporting different tailoring engines
    - Segmentation data is usually some of the largest data (of i18n data?) that clients at Google need to ship, so being flexible on when and which parts to ex-/include is important
    - The ICU4X DataProvider's design for examining and controlling which data is included during which phase (compile-time, load-time, run-time) is very relevant
    - It's clear in discussion of types of tailorings that flexibility up to
- Naming
    - BreakIterator vs. segmenter
    - ICU has BreakIterator
    - In localization industry, "segmentation" specifically means "break into sentence"
    - In Google, "segmentation" means word tokenization
    - ECMA-402 precedent is "segmenter" (Intl.Segmenter)
    - BreakIterator may have overlap in connotations in Rust due to the prevalence of the iterator interface
- Proposed TODO list / roadmap before merging to upstream `main` branch: https://github.com/aethanyc/icu4x/issues/1
    - Good to check in smaller pieces whenever ready
- Discuss pros and cons of: porting Rust crates for UAX 14, UAX 29 vs. writing code to read the ICU data for RBBI
    - Already discussed this week a tool in ICU that can export data for code point tries (& other structures) to make it easier to consume in ICU4X
    - Would not recommend re-implementing the ICU RBBI builder unless you actually have to
    - Maybe creating a reader for the ICU RBBI data is a good way to start initially, but eventually, it might be nice to have a builder implementation in the long-term to allow low-cost tailorings
    - Note: ECMA-402 supports random access (of what?) after significant discussion

    - We might want ICU4X in the long-term to have native builder code to allow self-contained code that minimizes ICU dependences that make it difficult for new ICU4X contributors
    - At no point are we creating dependencies on ICU in ICU4X. We are discussing a tool to export the ICU data in a usable way that is not tied to ICU code implementation details. And instead of moving away from ICU, moving towards ICU is a direction we should embrace
    - Main downside to reader for ICU data discussed so far: we want to have a tailoring engine to apply tailorings at runtime. What are the use cases for saving data size? There is already a lot of rule data required, so how much size would be incurred by implementing and shipping builder code, too?
        - Not sure where the balance lies
        - Maybe if there are 10s of KBs of rule data for multiple languages, then having a native implementation of a RBBI builder would be an overall savings, but I don't know the exact numbers to say if that's the case
- How does this tie in with SE Asian language segmentation using ML (LSTM model)?
    - The LSTM model incorporates breaking on spaces and words, whereas ICU's dictionary-based approach only works on the spans of text between stop characters/whitespace
- How much (percentage) size savings were gained by changing the ICU RBBI builder code's output FSM tables from using 16-bit words to 8-bit words where possible?
    - (see https://github.com/unicode-org/icu/pull/1100#issuecomment-610771298)
      These changes landed in ICU 68
      https://docs.google.com/spreadsheets/d/1v4YkY9MBptrqy9ger55CPITc9Ppm2krNTMDmgMjHF90/edit#gid=0
      Column H ("Rules") shows the size of the rule strings
    - This is the size of compiled resource at rest in the *.res file
    - Doesn't represent the size before compression
    - These file sizes still include the rule string. But the rule string could still be omitted. On the other hand, the rule string itself is not that big, so the impact of omitting may not be small
    - Total size of all rule strings is about 75 KB, and that is more than 25% of the entire RBBI data size, so excluding them would be useful for ICU4X
    - The rule strings exist to satisfy the public API that gives you the rule strings for a BreakIterator instance. But few people actually use that
    - Keep in mind that the compiled form can be bigger than source form (or oftentimes, this is the case -- the extra space requirement is a tradeoff that gives you faster runtime performance)
- In ICU4X, are we going to handle different encodings (UTF-8, UTF-16, Latin-1) in the input strings?
    - Firefox uses UTF-16 internally
    - For DOM, Mozilla also uses Latin1 in some cases. So would be nice to use that directly without conversion
    - So we would like to iterate on the inputs directly without having to convert encodings to just a single encoding (ex: UTF-8)
    - Note: break iterator indices will be done by code units, not by code points. Most languages provide indices into strings according to the code unit as determined by the encoding, not by code points

    - In Rust, you use code unit indices. The idea is a separate algorithm gives you code unit indices, because working on code points directly in the different encodings is slow
- What approach does ICU4X prefer: Rust native impl of segmentation, or reuse ICU BreakIterator data via porting reader code to Rust?
    - In ICU, when it comes to integration of the ML segmentation for SE Asian (LSTM model), we need to add a hard-limit for the length of backward direction traversal in the LSTM algorithm to prevent a memory overflow on very long strings.
        - Need to consider for ICU4X too
        - Perhaps read the backwards traversal first, then can use constant space memory when doing the forwards traversal next. Or symmetrically, can do the forwards traversal first and store it, then use constant space memory doing the backwards traversal next
    - Preference is to try to integrate the existing branch, review can generate further comments and revisions. In parallel, we can work on an approach that reads ICU data that requires creating support for CodePointTries in ICU4X
        - Only concern there is that the approach of a reader of the ICU RBBI data is so different from Rust native implementation that there may not be much overlap of the code in the 2 approaches
        - Counter-concern is the timeline for getting something working would be pushed out to wait for the longer-term solution
        - Another option to address the short-/medium-term timeline concern is to have a create that exposes an ECMA-402 API and can rely on uax14_rs and unicode-segmentation Rust crates as dependencies. So long as we don’t take on large amounts of code in ICU4X main branch that we don’t yet know for sure that we want to maintain long-term
    - Bonzai method for decision-making: evaluate success criteria, evaluate alternatives against success criteria, and prune potential effort according to the success criteria
    - One next step can be to evaluate the approach of a reader for ICU RBBI data structure output data files
        - for both this experiment and other properties work, someone will need to port the runtime code for the ICU code point trie to Rust https://github.com/unicode-org/icu4x/issues/508