Collation tailoring issues

Please add comments if you disagree with any of the following.

special reset positions always from root

ICU treats special reset positions like &[last variable] as always referring to the collation elements of those positions in the root collator. For example,

```
&[last variable]<x &[last variable]<y
```

puts y before x, although x was just tailored to be another variable after the last variable root item.

This is unlike normal tailoring rules where each builds on top of the state of all previous rules. However, it is much easier to implement than the alternative, and therefore I intend to keep it this way (and document it).

Same for

```
&[last primary ignorable]<<x &[last primary ignorable]<<y
```

etc.

Decision changed later. See http://unicode.org/cldr/trac/ticket/6070#comment:6

[last variable] vs. [maxVariable]

Similarly,

```
&[last variable]<x [maxVariable symbol] &[last variable]<y
```

will yield the same result for x and y as above, rather than putting y at the end of the symbols group.

This also matches the CLDR spec for the old reset-last-variable syntax which says “The value can be further changed by using the variable-top setting. This takes effect, however, after the rules have been built, and does not affect any characters that are reset relative to the last-variable value when the rules are being built.”

Reminder: The plan is still to deprecate the syntax

```
&'$' = [variable top]
```

so I am not planning to support it in my new implementation.

The maxVariable setting, which replaces that, will then also apply only “after the rules have been built”.

suppressContractions

The LDML spec does not specify any of the following, and the ICU documentation does not mention suppressContractions.

suppress when?

ICU collects the union of all [suppressContractions] sets and at the end suppresses them all together. I think this is incorrect. For example,

```
[suppressContractions [ə]] &ə=x
```

currently makes x sort primary-greater than ə because, while processing the reset and tailoring x, the root contraction for ə has not been suppressed yet.

Also, prefixes and contractions are stored together with the data for their originating code points. While adding the first tailored mapping for a code point, all of its contexts need to be copied from the root data, unless they are suppressed.

I intend to suppress the relevant mappings right when this command is encountered, so that those mappings are gone from the tailoring state seen by future rules. This is consistent with the semantics of other tailoring rules that affect mappings. (As opposed to parametric settings.)

suppress prefixes?

ICU suppresses not just contractions but also prefixes. For example, this

```
[suppressContractions [·]]
```

suppresses the root collator’s mappings for l|· and L|·.

I think this is fine, and intend to keep it that way and document it as suppressing all contextual mappings that originate with any “suppressed” code point. (If code point c is in the set, it suppresses all s and all p|s where c is the first code point of s.)

Note that a prefix item like L|· is similar to a contraction of L·, but in order to suppress the former you need to list the middle dot, and in order to suppress the latter you need to suppress the L. Note also that the CLDR root replaces the related DUCET contractions with prefix mappings. For suppressContractions to be useful, users need to have a good idea of what specific mappings there are.

suppress tailorings?

While special commands are usually placed at the start of the rules, the syntax allows them

anywhere, and they will naturally come in the middle when concatenating rules (as we suggest users to do sometimes) or when using [import] (as we will want to do with search and EOR tailorings).

Example:

```
&z<ch<əж [suppressContractions [cə]]
```

I intend to make this suppress (a) the root contraction ə, (b) the tailored contraction ch, and (c) the tailored contraction əж — because I think that makes for reasonably consistent semantics. It also eliminates the need to track in the builder data structures which contractions are unchanged from the root collator vs. added from tailoring rules.

- Note that when tailoring əж the builder needed to copy all contextual mappings originating with ə, including ə.
- If we wanted to only suppress root contractions, but we had tailored ə, we would need to have enough state to know that and keep this tailored contraction.
- This would be the only way to un-tailor previously-tailored contractions and prefixes when concatenating or importing rules.

syntax for quaternary relations

We decided to keep the ICU syntax unchanged, and use <, <<, <<<, = in CLDR. We need new syntax for quaternary relations, to be used in the Japanese tailoring where that would replace the [hiraganaQ on] setting (which we agreed to deprecate).

I intend to add syntax <<<< (and <<<<* for the compact version).

- pro: natural extension of the other syntax we have agreed to use in CLDR
- con: it also extends the contradiction that weaker relations look bigger

Note: In my previous proposal for more radical syntax additions, I had proposed :: for the quaternary relation. That does not make for a natural extension, and it also has the same weaker-is-bigger problem (it uses 4 dots), although it is visually less conspicuous.

I will not add support for &[before 4]x — it will only be possible to tailor quaternary-after.

Note: There will be a limitation for at most 3 quaternary relations in a row, since I am using only 2 bits to store quaternary weights. It will also not be possible to create quaternary CEs (that is, CEs that are tertiary ignorable but have non-zero quaternary weights).

how to compute case bits

background

LDML spec 3.13 Case Parameters

- In the root collation (FractionalUCA.txt), each collation element has one bit for uncased/lowercase/small Kana vs. uppercase/other Kana. We never store “mixed case” in root.

- In a tailoring, case bits are set algorithmically by the builder, not specified via rules. We have three values, with a middle “mixed” case. (2 bits, one value reserved for use in sort keys)
- The current spec is not yet implemented. (ICU ticket 9337)
- The current spec computes case bits only for the first CE of a tailored expansion, and we would have to maintain its detailed derivation to keep it in sync with the root case bits.
- Mark and I had planned to revisit the case bits algorithm, maybe using data from the decomps.txt UCA file. I believe that since the root collator data was constructed from the same source data, we can just work with what we have in the root collator anyway.
- Almost all Cased characters have primary (non-ignorable) CEs. (Except for U+0345 combining Ypogegrammeni which is Lowercase.) All Uppercase characters have primary CEs.

proposed algorithm

1. Look up the tailored string’s root CEs. N=number of primary root CEs.
2. Determine the number and type (primary vs. weaker) of CEs a tailored string maps to. M=number of primary tailored CEs.
3. If N<=M (no more root than tailoring primary CEs): Copy the root case bits for primary CEs 0..N-1.
    a. If N<M (fewer root primary CEs): Clear the case bits of the remaining tailored primary CEs. (uncased/lowercase/small Kana)
4. If N>M (more root primary CEs): Copy the root case bits for primary CEs 0..M-2. Set the case bits for tailored primary CE M-1 according to the remaining root primary CEs M-1..N-1:
    a. Set to uncased/lower if all remaining root primary CEs have uncased/lower.
    b. Set to uppercase if all remaining root primary CEs have uppercase.
    c. Otherwise, set to mixed.
5. Tertiary-ignorable CEs 0.0.0 must get ignorable-case=lowercase bits.
6. Tertiary CEs 0.0.t must get uppercase bits.

For an extension, just copy the CEs for the extension string, including case bits. (This is different from expansions.)

This is easy to implement, does not require additional data, and is always consistent with the root collator case bits.

It also sets reasonable case bits in expansions. For example, &ae=cH yields two collation elements with [lower, upper] case bits, rather than the current spec’s mixed case on the first CE. Document this as yet another difference between &ae=cH and &a=cH/e.