**Fractional Plurals Design Doc (draft)**

Live doc: <http://goo.gl/AJg6v>

ICU (and CLDR) did not originally support plural categories for fractions. So we would correctly say “1 dollar”, “10 dollars”, but fail to correctly show “1.00 dollars”. As it turns out, plural fractions are quite varied; almost as varied as integer fractions. This document describes the changes needed to implement this in CLDR/ICU.

There are a few areas with open issues: see the “Proposal:” sections below.

***Contents***

[Plural Rules](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.50wzyhptyba9)

[Spec Changes](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.cc1j5ck9r6xb)

[Definition](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.qs3sxlvq8zrd)

[Examples](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.x1l1gekjccjq)

[Proposal: Base Values](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.r1ablhbb505f)

[Ordering of rules](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.kcxzaxyrc5mf)

[Future Syntax?](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.gvkf8wthj10)

[In vs is](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.il8wwaxs4fn)

[Modulo](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.ixvf8kwien6)

[CLDR Data](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.6ljpwiaeh353)

[ICU Implementation](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.lt0xs22vpw8v)

[PluralRules](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.a9anjnnh89r7)

[NumberInfo](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.52fd8tkb5l7t)

[Serialization](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.fikgvt94fij1)

[Equality](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.gw4ykxb6hjpt)

[Samples](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.quh5nbu2wewb)

[Proposal: Samples](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.tod2jlvaulr6)

[Number/MessageFormatting](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.7teg3nsladsl)

[Problems](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.mztjrq8gtjgk)

[Proposal: Formatting (Java)](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.8v9xucvithtc)

[Keywords](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.ypal2mls0sms)

[API Problems](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.b0t2togug50q)

[Proposal](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.ws0ed5awi9nf)

**Plural Rules**

The first part of the support are new plural rules. We first gathered some raw information from a large number of languages, where we used “minimal pair phrases” coupled with different combinations of numbers: 1, 1.0, 1.00, 1.1, 2, 2.0, 2.1, etc. These numbers were computed by taking combinations of pairs of samples from the existing integer categories for each language. We also tried different combinations of trailing zeros, since those make a difference for some languages. On that basis, we came up with draft rules, which we used to generate further samples, and check and refine the results.

**Spec Changes**

The changes to [Language Plural Rules](http://www.unicode.org/reports/tr35/tr35-numbers.html#Language_Plural_Rules) are expansions of the syntax to allow for fractions, and to test different features of a number.

**ICU4J 51.2 rule syntax (from PluralRules.java javadoc) (probably unchanged for several years)**

 rules         = rule (';' rule)\*

 rule          = keyword ':' condition

 keyword       = <identifier>

 condition     = and\_condition ('or' and\_condition)\*

 and\_condition = relation ('and' relation)\*

 relation      = is\_relation | in\_relation | within\_relation | 'n'

 is\_relation   = expr 'is' ('not')? value

 in\_relation   = expr ('not')? 'in' range\_list

 within\_relation = expr ('not')? 'within' range\_list

 expr          = 'n' ('mod' value)?

 range\_list    = (range | value) (',' range\_list)\*

 value         = digit+

 digit         = 0|1|2|3|4|5|6|7|8|9

 range         = value'..'value

**“Small extension” of the syntax, agreed in 2013-aug-07 ICU meeting:**

 rules         = rule (';' rule)\*

 rule          = keyword ':' condition

 keyword       = <identifier>

 condition     = and\_condition ('or' and\_condition)\*

 and\_condition = relation ('and' relation)\*

 relation      = is\_relation | in\_relation | within\_relation | 'n'

 is\_relation   = expr 'is' ('not')? value

 in\_relation   = expr (('not')? 'in' **| '=' | '!='**) range\_list

 within\_relation = expr ('not')? 'within' range\_list

 expr          = **(**'n' **| 'i' | 'f' | 't' | 'v')** (**(**'mod' **| '%'**) value)?

 range\_list    = (range | value) (',' range\_list)\*

 value         = digit+

 digit         = 0|1|2|3|4|5|6|7|8|9

 range         = value'..'value

Plus:

* Accept only non-uppercase ASCII, document syntax limitation for the future.
* keywords: restrict to only [a-z]+
* keywords: allow any [a-z]+ (do not forbid syntax words like “within”, do not limit to CLDR plural keywords)

**Just for temporary reference: Mark’s original proposal for more extensively modified rule syntax**

 \* rules         = rule (';' rule)\*

 \* rule          = keyword ':' condition

**\* keyword       = [a-z]+ ([\_] [a-z]+)\***

 \* condition     = and\_condition ('or' and\_condition)\*

 \* and\_condition = relation ('and' relation)\*

**\* relation      = not? expr not? rel not? range\_list**

**\* expr          = ('n' | 'i' | 'f' | 'v' | 't') (mod value)?**

**\* not           = 'not' | '!'**  // There cannot be more than one “not” term.

**\* rel           = 'in' | 'is' | '=' | 'within'**

**\* mod           = 'mod' | '%'**

 \* range\_list    = (range | value) (',' range\_list)\*

 \* value         = digit+

 \* digit         = 0|1|2|3|4|5|6|7|8|9

 \* range         = value'..'value

**Definition**

The values of expressions containing n, i, f, t, v, ~~, and j~~ values for a given source number are defined as follows:

1. Let BK be the base keyword:  one of ('n' | 'i' | 'f' | ’t’ | 'v' ~~| 'j'~~) as defined in the table below.
2. Let the base value BV be computed from absolute value of the original source number as in the table below.
3. Let R be false when the comparison contains ‘not’.
4. Let R be !R if  the comparison contains ‘within’ and the source number is not an integer.
5. If there is a module value MV, let BV be BV - floor(BV/MV).
6. Let CR be the list of comparison ranges, normalized that overlapping ranges are merged. Single values in the rule are represented by a range with identical <start,end> values.
7. ~~If BK is ‘j’ and v ≠ 0, then return !R.~~
8. Otherwise iterate through CR: if starti ≤ BV ≤ endi then return R.
9. Otherwise return !R.

|  |  |
| --- | --- |
| **BK** | **BV** |
| n~~, j~~ | source as a double float |
| i | integer digits of |n| (as a long integer) |
| f | *visible* fractional digits in |n| (as a long integer). |
| t | f with trailing zeros removed |
| v | number of *visible* fraction digits in |n| (as an integer). |

Rules are currently only defined for positive source values, and for integer modulo values.

**Examples**

Examples of the base values are in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **i** | **v** | **f** | **t** |
| **1.0** | 1 | 1 | 0 | 0 |
| **1.00** | 1 | 2 | 0 | 0 |
| **1.3** | 1 | 1 | 3 | 3 |
| **1.30** | 1 | 2 | 30 | **3** |
| **1.03** | 1 | 2 | 3 | 3 |
| **1.23** | 1 | 2 | 23 | 23 |

~~The keyword j is special: it restricts matching to where there are no visible fractions. So “j is 3” means that “n is 3 and v is 0”. Thus:~~

* ~~“j is 3” fails if v ≠ 0 (eg~~ **~~true~~** ~~for 3, but~~ **~~false~~** ~~for 3.1 or 3.0);~~

~~As always, ‘not’ means the inverse. So “j is not 3” means that “n is not 3 or v is not 0”~~

* ~~“j is not 3” succeeds if v ≠ 0  (eg~~ **~~false~~** ~~for 3, but~~ **~~true~~** ~~for 3.1 or 3.0);~~

**Proposal: Base Values**

1. There are languages that care about the fractions, but don’t care about the trailing zeros. For example, X.1 is significant, but always treated like X.10 and X.100, etc. We currently don’t handle those languages well. For that, I propose that we add yet another base value:

|  |  |
| --- | --- |
| t | *visible* fractional digits in source (as a long integer) *with trailing zeros removed.* |

Here is the table difference:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **i** | **v** | **f** | **t** |
| **1.0** | 1 | 1 | 0 | 0 |
| **1.00** | 1 | 2 | 0 | 0 |
| **1.3** | 1 | 1 | 3 | 3 |
| **1.30** | 1 | 2 | 30 | **3** |
| **1.03** | 1 | 2 | 3 | 3 |
| **1.23** | 1 | 2 | 23 | 23 |

1. **Open Issue:** In my first cut, I changed the constants so that they could also be fractional. That is, you could say “n is 3.2” (that used to be illegal). As it turns out, we do not need that, so we could remove it, making the code slightly simpler. **REMOVED**
2. **~~Open Issue:~~** We could simplify slightly if we made **n** be the integer form; that is, it would just have the value of **i** above, and we would get rid of **i**. Because the **v**, **f** (and **t**) values are available, we don’t really need the double value of **n**. That would make the internals slightly simpler, and have slightly simpler rules for the user. **NOT DONE (compat)**
3. **Open issue:** This may not be the most natural interpretation; might be better to have “j is not 3” means that “n is not 3 and v is 0”**. RETRACTED J**

**Additional restrictions**

Rework all the following:

1. A keyword cannot occur twice, must be executable in any order (except 'other', which must be last).
2. Samples are not computed if any rule has @integer or @decimal. In that case, those samples are presumed to be valid. If the samples are bounded (no “...”), then they must be complete. If the samples are missing, then there are no possible samples for that type for that keyword.
3. ‘other’ must contain at least one unbounded set of samples (integer or fraction). Samples will not be correct unless included explicitly.
   1. If it has any elements in either @integer or @decimal, then that set must be unbounded.
4. Semantic changes:

PluralRules have a number of methods that depend on which items are matched by a keyword (getSamples, getKeywordStatus, etc) . In particular: what are examples of such matches; is the set of matches empty, unique, bounded, or infinite; and if not infinite, what are they?

Beforehand, fractions didn't really work at all. When we added them this cycle, I found that the best compatibility appeared to result from making all of those methods above just test for the INTEGER cases. Earlier, I had the methods return the union of the results. That, however, returns the wrong answer when people are using integers, which is the most common use case. So, I added an optional parameter to get the DECIMAL cases instead, and the option-less methods just tests the INTEGER case.

Migration:

                Collection<Double> list = rules.getSamples(keyword);

=>

                Collection<Double> list = rules.getSamples(keyword);

                if (list.size() == 0) {

                    // if there aren't any integer samples, get the decimal ones.

                    list = rules.getSamples(keyword, SampleType.DECIMAL);

                }

**Ordering of rules**

The rules are often stated in a clumsy way right now, because each set of rules must be self-contained. Yet to evaluate the rules, we have to walk through the categories anyway until we find a match. We agreed to change so that the order of that walk is predictable (zero, one, two, few, many, other, then any others in alphabetical order) allowing us to simplify the rules.

**Future Syntax?**

The syntax is a bit clumsy, even for programmers. I’m capturing here some ideas for simplifying the syntax a bit.

**In vs is**

The syntax is not really English, but also not consistent.

x in 3, 5-9

x *not* in 3, 5-9

x is 3

x is *not* 3

Probably simplest to just treat ‘in’ and ‘is’ as synonyms, and allow ‘not’ in any position: before/after the subject or relation: eg, “… and not x in 3, 5-9”. It would also, I think be clearer to add =, ≠, so you could write x ≠ 3 instead of x is *not* 3.

**Modulo**

We have rules like:

n mod 10 is 1 and n mod 100 is not 11

It would be simpler (and in keeping with the “English style” syntax) if we could phrase these rules like:

n endswith 1 and n not endswith 11

Note that leading zeros are significant: “n endswith 01” is *not* the same as “n endswith 1”.

**CLDR Data**

As a reminder, the plural categories are used to format messages with numeric placeholders, expressed as decimal numbers. The fundamental rule for determining plural categories is the existence of *minimal pairs*: whenever two different numbers *require* different versions of the same message, then the numbers have different plural categories. See [Determining-Plural-Categories](http://cldr.unicode.org/index/cldr-spec/plural-rules#TOC-Determining-Plural-Categories).

This happens even if nouns are invariant; *even if* all English nouns were invariant (like “sheep”), English would still require 2 plural categories. For example:

1. 1 sheep **is** here.
2. 2 sheep **are** here.

The key is “requires”. English does not have a separate plural category for “zero”, because it does not require a different message for “0”. For example, the same message is used below, with just the numeric placeholder changing.

1. You have 3 friends online.
2. You have 0 friends online.

Because it is quite common that it is more natural to express 0 messages with a negative (“None of your friends are online.”), pluralized message APIs offer the ability to specify the 0 case explicitly; developers should use this whenever 0 might occur in a placeholder.

According to the data we have gathered, the following plural fraction behaviors occur:

|  |  |  |  |
| --- | --- | --- | --- |
| **Behavior** | **Description** | **Example** | **Sample Rules…** |
| ***Base*** | The fractions are ignored; the category is the same as the category of the integer. | 1.13 has the same plural category as 1. | i is 1 |
| ***Separate*** | All fractions by value are in one category (typically ‘other’ = ‘plural’). | 1.01 get the same class as 9  1.00 gets the same category as 1. | n is 1,9 |
| ***Visible*** | All *visible* fractions are in one category (typically ‘other’ = ‘plural). | 1.00 and 1.01 get the same category as 9. | j in 1,9 |
| ***Digits*** | The visible fraction determines the category. | 1.13 gets the same class as 13. | f mod 100 is 13 |

There are also variants of the above: for example, short fractions may have the Digits behavior, but longer fractions may just look at the final digit of the fraction. There may be other patterns in languages that we’ve not yet gathered data for.

For the current CLDR24 rules, see: [Language plural rules](http://www.unicode.org/repos/cldr-aux/charts/24/supplemental/language_plural_rules.html).

**ICU Implementation**

The ICU plurals implementation is used by CLDR, so had to be changed before the data submission could be made. There are a few outstanding issues.

For reference, here is the old (release) documentation: [PluralRules](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html).

**PluralRules**

The ICU trunk code supports the syntax changes described above. There are some additional ICU APIs, but they are all marked @internal for now.​ These were necessary to gather and compare the data for different fractional forms. Some of them could be useful to normal clients, so we can look at making some of them public, but later on.

**NumberInfo**

I added the class NumberInfo, because we need a way for people to supply numbers, and for us to pass around numbers internally while retaining the exact visible fraction value. There are some convenience methods to make it simple to get a default NumberInfo from a double.

NumberInfo extends Number, and can thus easily be converted to a double. It is currently @internal. At some point we will want to make it public, probably with a better name.

**Serialization**

It was impossible to make serialization work across versions, so the committee agreed to allow a break. See <http://bugs.icu-project.org/trac/ticket/8474>. Since only the string format is stored, it will be much easier to maintain compatibility in the future. As it turns out, the simplest approach was to create a serial proxy handler, which passes the string format of the rule. That appears to be the way to retain final fields in the original object. As a byproduct the serialization now takes about 1/4 the space, although that was not a goal.

**Equality**

Another byproduct of fraction support was equals(). The previous implementation could iterate through all the possibilities, up to the 'repeat limit'. That is no longer feasible with fractions, so the code now just tests whether the normalized rules for each keyword are identical. (We punt in the same way for collation, etc.) That means, however, that the following rules are not recognized as equal:

1. "n in 1..2, 3..4"
2. "n in 3..4, 1..2"
3. "n in 1..4"
4. "n in 1..2 or n in 3..4"

While the ultimate goal would be that two PluralRules that always returned the same select() values would be equal, that would be a fair amount of work.

We could do some simple normalization of range lists on parsing to catch the equality between cases like 1-3. It would be simpler code to just enforce some stronger conditions when parsing. For example, we could easily test ranges and throw an exception in cases like:

....a, b.... where a >= b, or where a + 1 == b and operation is not "within"

My inclination is to just file a bug for some of this as possible future work or add a TODO to the code.

**Note**: we should document that the input format may differ from the output. For example, we could canonicalize the order at some point, and combine statements. For example:

two: v is 0 and n is 1..2 and n is not 1

→

two: n is 2 and v is 0

**Samples**

Samples turn out to be a significant issue: see [API Problems](https://docs.google.com/document/d/155ZJOHtOgnm8P80TDL8QGfNZ-wNoqsRNRGHRfJB4JGs/edit#heading=h.b0t2togug50q).

**Number/MessageFormatting**

There is an initial cut at number formatting, including units (but not yet currencies). It is a bit of a hack.

**Problems**

We need to know the visible fractions in order to pick the plural category. That is done right now by returning extra information in the FieldPosition. Internally, it is a pain to do because we first compute the DigitList, but it does not contain all the digits. The leading and trailing zeros are added when the number is formatted, so there is special code there to observe when a trailing zero is added. But that means going all the way through the formatting process.

Moreover, in order to format units (including currencies), we need to select the right pattern according to the plural category. Currently that is done by formatting twice. We first use the NumberFormat to format the number, finding out the visible fractions. We then pick the pattern, and format again. This is ugly and inefficient.

MessageFormatting has a further problem. Suppose that we have a format like the following

MessageFormat msgFmt = new MessageFormat(

     "{num\_files, plural, " +

     "=0{There are no files on disk \"{disk\_name}\".}" +

     "one{There is # file on disk \"{disk\_name}\".}" +

     "other{There are {num\_files,number,0} files on disk \"{disk\_name}\".}}",

     ULocale.ENGLISH);

The formats used by the **#** and **{num\_files...}** have to be consistent. A bad case would be with the number 1.0 where the format in the ‘other’ case has no decimals, causing it to need to use the ‘one’ category, but the format in the ‘one’ category does have decimals, pushing the number to the ‘other’ case! Markus and I discussed this, and came to the conclusion that (a) resetting the decimals doesn’t happen much anyway, and (b) it was ok to put some burden on the programmer to not supply strange patterns.

For communicating the information, I also created a subclass of field position, for which Yoshito had the following comment. However, I think if we restructure as in the proposal below, we no longer need that.

* (Yoshito) I think FieldPosition is somewhat old approach to carry field information. When such information is necessary for multiple fields, extending java.text.Format.Field is the way to go. My understanding is that AttributedCharacterIterator is providing more comprehensive support and replacing FieldPosition.

**Proposal: Formatting (Java)**

1. We change DigitList so that it takes all the formatting constraints into effect. That is, a DigitList will include *every single digit* that will appear in the output, including leading and trailing zeros.
2. Moreover, we add **internal** APIs to NumberFormat to:
   1. create a DigitList from a number (of whatever form).
   2. format a DigitList.
   3. Note: internal to the code, formatting a number will change to just calling these two in sequence.
3. For MessageFormat, we use the decimals for the pattern used in the ‘other’ case, for the first instance of # if there are multiple.
   1. We document that it is the programmer’s responsibility to ensure that there are no (bizarre) inconsistencies.

That way, for formatting units, currencies, and messages, we can create a DigitList externally, and ask it for fraction info so that we can get the right plural category. We then pick the pattern we want, and pass in the DigitList when formatting that pattern, so that we don’t have to create the DigitList twice.

**Open Issue: we need to figure out the C API to deal with this.**

**Keywords**

We agreed to revert a couple of changes:

1. The keywords in ICU will be ordered in the order that they are created in. (As before.) CLDR, however, will always create them in canonical order: **zero, one, two, few, many, other**.

I looked into other issues, and I don’t think we can actually remove the following restrictions, or at least it would be a lot of work:

1. There can be only one rule per keyword, eg “a: n is 1; a: n is 5” is illegal because ‘a’ occurs twice.
2. Rules must be self-contained, eg “a: n is 1; b: n is 1..5” is illegal, because both the rule for ‘a’ and the rule for ‘b’ match ‘1’.

**API Problems**

We agreed to move the samples-finding code to CLDR, and simply store the data for the samples and return it via API. It turns out to be not completely simple. The problem is that it is not trivial to compute what all the possible samples are for a given rule, especially given fractions.

There are two main goals for these APIs. These are used by CLDR Survey Tool (and CLDR charts), but also at runtime in other translation UIs—such as Google’s.

1. Show some samples for each keyword.
2. Determine when a keyword is triggered for only one value (it is “unique”). In that case the message doesn’t require a “{0}”.
   1. Arabic has “one: n is 1”, for example. See [Arabic plural rules](http://www.unicode.org/repos/cldr-aux/charts/24/supplemental/language_plural_rules.html#ar).
   2. It doesn’t need an explicit number since that is clear from the grammatical category.
3. Determine when a keyword is unused. In that case, no message needs to be supplied.
   1. For example, Czech has “few: v is 0 and i in 2..4”.
   2. If there are explicit values for =2, =3, and =4, then the message for “few” is never used.
   3. If there is an offset of 4, then the message for “few” is never used.
   4. For two digit numbers (without leading zero), “few” can never match. This is the case for compact decimals with the pattern 00 (eg 10..99).
   5. Moreover, in some instances, the developer knows that no decimal values will appear: the user would never see “You have 1.3 friends”. Thus for Czech, “many” would be unused with no decimals.
4. For 3c, we need to compute whether a keyword has any matching values, given a specific number of digits without leading zero. This is needed for Compact Decimals.

All of this needs to be updated for fractions. And bear in mind that the number and list of visible digits affects the results. For example,

* in some languages a visible fraction ending in “1” makes the number singular: ***2 books*** but ***2.1 book***.
* in some languages, a visible fraction ending in zero makes the number plural: ***1 book*** but ***1.0 books***.

We currently have API (listed below) that tells us

1. if the keyword has a unique matching value, and what it is ([**getUniqueKeywordValue**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getUniqueKeywordValue(java.lang.String))).
2. if the keyword has finite matching values, and what they are ([**getKeywordStatus**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getKeywordStatus(java.lang.String,%20int,%20java.util.Set,%20com.ibm.icu.util.Output)), [**getAllKeywordValues**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getAllKeywordValues(java.lang.String))).

(Note that [**getUniqueKeywordValue**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getUniqueKeywordValue(java.lang.String)) is superfluous, since the combination of [**getKeywordStatus**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getKeywordStatus(java.lang.String,%20int,%20java.util.Set,%20com.ibm.icu.util.Output)) + [**getAllKeywordValues**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getAllKeywordValues(java.lang.String)) can be used instead. Also, if [**getSamples**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getSamples(java.lang.String)) returned the same value as [**getAllKeywordValues**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getAllKeywordValues(java.lang.String)) for BOUNDED keywords, we don’t need [**getAllKeywordValues**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getAllKeywordValues(java.lang.String)).)

The situation is complicated by the fact that we didn’t really support fractions up until now, yet the API is phrased in terms of Doubles. Yet Doubles are insufficient, since we have to know the visible fractions, which Double alone doesn’t tell us. We have a new class NumberInfo (@internal for now, and which we might rename), which does have the additional information as to the visible fractions.

Here is the relevant current released API. All are @stable except for [**getKeywordStatus**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getKeywordStatus(java.lang.String,%20int,%20java.util.Set,%20com.ibm.icu.util.Output)):

|  |  |
| --- | --- |
| [**Collection**](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html?is-external=true)<[**Double**](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html?is-external=true)> | [**getAllKeywordValues**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getAllKeywordValues(java.lang.String))([**String**](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html?is-external=true) keyword)  Returns **all** the values that trigger this keyword, or null if the number of such values is unlimited. |
| [**PluralRules.KeywordStatus**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.KeywordStatus.html) | [**getKeywordStatus**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getKeywordStatus(java.lang.String,%20int,%20java.util.Set,%20com.ibm.icu.util.Output))([**String**](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html?is-external=true) keyword, int offset, [**Set**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html?is-external=true)<[**Double**](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html?is-external=true)> explicits, [**Output**](http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Output.html)<[**Double**](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html?is-external=true)> uniqueValue)  **@provisional**  Find the status for the keyword, given a certain set of explicit values. |
| [**Collection**](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html?is-external=true)<[**Double**](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html?is-external=true)> | [**getSamples**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getSamples(java.lang.String))([**String**](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html?is-external=true) keyword)  Returns a list of values for which select() would return that keyword, or null if the keyword is not defined. |
| double | [**getUniqueKeywordValue**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getUniqueKeywordValue(java.lang.String))([**String**](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html?is-external=true) keyword)  Returns the unique value that this keyword matches, or [**NO\_UNIQUE\_VALUE**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#NO_UNIQUE_VALUE) if the keyword matches multiple values or is not defined for this PluralRules. |

The KeywordStatus values are all **@provisional**:

|  |
| --- |
| [**INVALID**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.KeywordStatus.html#INVALID)  The keyword is not valid for the rules. |
| [**SUPPRESSED**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.KeywordStatus.html#SUPPRESSED)  The keyword is valid, but unused (it is covered by the explicit values). |
| [**UNIQUE**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.KeywordStatus.html#UNIQUE)  The keyword is valid, used, and has a single possible value (before considering explicit values). |
| [**BOUNDED**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.KeywordStatus.html#BOUNDED)  The keyword is valid, used, not unique, and has a finite set of values. |
| **U**[**NBOUNDED**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.KeywordStatus.html#UNBOUNDED)  The keyword is valid but not bounded; there indefinitely many matching values. |

**Proposal**

Document that the above API is only well defined for integers (since up until now, only integers worked).

Add the following (@internal for now) single routine that bundles together all the needed functionality.

|  |
| --- |
|  |

|  |  |
| --- | --- |
| [**PluralRules.KeywordStatus**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.KeywordStatus.html) | [**getKeywordStatus**](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/PluralRules.html#getKeywordStatus(java.lang.String,%20int,%20java.util.Set,%20com.ibm.icu.util.Output))([**String**](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html?is-external=true) keyword, int offset, [**Set**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html?is-external=true)<[**Double**](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html?is-external=true)> explicits, int integerDigits, boolean hasDecimals, Set<Number> samples)  Find the status for the keyword, given a certain set of explicit values and offset.  Returns in **samples** a list of numeric values for which select() would return that keyword, or sets to empty if the keyword is not defined. If the KeywordStatus is BOUNDED or UNIQUE, then returns **all** the values that trigger this keyword.  For **integerDigits** == 1..4, only considers value that would have the specified number of integer digits without leading zero. Example: 2 would be for values 10..99. The value **integerDigits** == 0 means any number of integer digits. An exception will occur if **integerDigits** is negative or greater than 4.  If **hasDecimals** is false, then only considers/returns values with no visible decimal fractions. For example, for Czech, “few” + hasDecimals=false would return SUPPRESSED & samples = EMPTY. |

Note that NumberInfo *isa* Number, and will be returned in samples from this routine.

½ apple →