Draft for Public Review

CLDR Person Name Formatting

LDML Part nn: Person Name (name structure, formats, sorting)

Version	2021-11-01
Editors	Mark Davis, Peter Edberg, Rich Gillam, Alex Kolesnichenko, Mike McKenna, with Manuel Coltorti, Peter Constable, Kristi Lee, and other CLDR Committee members
Timeline	Oct 14/15, present at Unicode conference Nov 01, PRI … refine and implement … Apr 01, 2022, CLDR v41 released with spec, phase 1 set of locale data, sample code, generated charts

Summary

CLDR Ticket: https://unicode-org.atlassian.net/browse/CLDR-13722

This document describes a proposed addition to the CLDR LDML specification to support the formatting of personal names, and uses data and structure that are consistent with that specification.

Problem

The content and structure of personal names can vary widely from region to region and locale to locale. This proposed addition to CLDR is based on review of current standards that exist in LDAP, hcard, and HTML as well as various commercial implementations.

A review of real world name examples, ranging from mononyms in Indonesia to patronymic and matronymic names in Iceland, Spain and Portugal to Arabic ancestral naming practices was also conducted.

The formatting patterns consider name usage in different legal, business, familial and formality contexts.

Goals

The primary goal of this effort is to provide a consistent method to format name objects considering various context and usage attributes. The initial proposal is built off the most ubiquitous name object structures found in LDAP, W3C and “contact” data commonly found in smartphones and address book apps. Minor additions to those object structures were made to accomodate known issues in large population groups, such as the need for a second surname in spanish-speaking regions and the common case of four names in Arabic speaking locales.

What this proposal includes

This proposal includes specifications for

- Structure of names, composed of name parts such as prefix, given name, middle name, surname, etc.
- Format of names in various contexts.
- Contexts considered are various lengths, different formats based on formality, and formats based on usage such as when referring to a person, addressing a person, or sorting names. These contexts could be extended in the future, such as for sports or bibliographic references.

What is out of scope for this proposal

The following features are out of scope for this proposal:

- Linguistic aspects when names change depending on context, for instance when the ending of a name may change when applied to a verb in some languages.
- Context-specific cultural aspects, such as when to use “-san” vs “-sama” when addressing a Japanese person.
- Differentiation between surnames and surname affixes such as “van”, “de”, etc. as in “van Gogh” vs “van” + “Gogh” (see: Tussenvoegsel)
- Validations as to which fields are required, and what encoding is allowed.
- Combining of names across locales, such as multi-cultural names in Hong Kong "Jackie Chan Kong-Sang”
- Parsing of names, such as identification of name parts.
- Parsing out the components of a name in a string, such as surname prefixes (Tussenvoegsel in Dutch) or differentiating between honorifics (Mr.,Hon.) , titles, generation indications (Jr.) and professional titles (MD). Even a seemingly simple name like "Mary Beth Estrella" could conceivably be any of the following

given	**middle**	**surname**	**surname2**
Mary	Beth	Estrella	
Mary Beth		Estrella	
Mary		Beth Estrella	

Mary		Beth	Estrella

A subsequent version of this document may add additional structure for assisting with other tasks, including:

- Enhanced formatting (such as handling special initial features for certain languages)
- Sorting (such as ignoring Tussenvoegsel)
- Parsing (separating out parts of a full name, such as the given vs surname)

The CLDR formatting is targeted at a single “name object” for a person. If multiple formatted names are needed, such as in different scripts or with alternate names, or pronunciations (eg kana), the presumption is that those are separate person name objects.

Status

This document is currently in the review period and is open to review and comment by the public.

Contents

1. Overview: Person Name Element, Formatting Information

```
<!ELEMENT personNames ( nameOrder*, personName+ ) >
```

The LDML top-level `<personNames>` element contains information regarding the formatting of person names, and the formatting of person names in specific contexts for a specific locale.

The `<nameOrder>` element is optional, and contains information about selecting patterns based on the order of elements in name. For more information see [XXX].

The `<personName>` element contains the format pattern, or `<namePattern>`, for a specific context and is described in Section 2 Person Name Elements.

The `<namePattern>` elements are described in Section 3 Person Name Format Patterns.

2. Person Name Elements

```
<!ELEMENT personName ( namePattern+ ) >
```

```
<!ATTLIST personName length LENGTH_VALUE #IMPLIED >
```

- `LENGTH_VALUE` is a space delimited list of `( long | medium | short | monogram | monogram-narrow )`

```
<!ATTLIST personName usage USAGE_VALUE #IMPLIED >
```

- `USAGE_VALUE` is a space delimited list of `( addressing | referring | sorting )`

```
<!ATTLIST personName style STYLE_VALUE #IMPLIED >
```

- `STYLE_VALUE` is a space delimited list of `( formal | informal )`

```
<!ATTLIST personName order ORDER_VALUE #IMPLIED >
```

- `ORDER_VALUE` is a space delimited list of `( surnameFirst | givenFirst )`

The `<personName>` element has attributes of `length`, `usage`, `style`, and order, and contains one or more `<namePattern>` elements.

- For each attribute, there must be at least one attribute value, no value can occur twice, and order is not important (but the canonical form is alphabetical). Thus style="informal informal" is invalid, as is style="".
- A missing attribute is equivalent to a list of all valid values for that attribute. For example, if style=... is missing, it is equivalent to style="formal informal".

```
<!ELEMENT namePattern ( #PCDATA ) >
```

A `namePattern` contains a list of fields enclosed in curly braces, separated by literals, such as:

```
<namePattern>{surname}, {given} {middle}</namePattern>
```

would produce as output "*Kennedy, John Fitzgerald*"

Person Name Object

The information that is to be formatted logically consists of a data object containing a number of fields. This data object is a construct for the purpose of formatting, and doesn't represent the source of the name data. That is, the original source may contain more information. The name object is merely a logical 'transport' of information to formatting; it may physically consist of, for example, an API that fetches fields from a database.

Note that an application might have more than one set of name data for a given person, such as data for both a legal name and a nickname or preferred name. Or the source data may contain two whole sets of name data for a Russian person, one in Cyrillic characters and one in Latin characters. Or it might contain phonetic data for a name (commonly used in Japan). The additional application-specific information in name data is out of scope for this document. Thus more than one name object can be extracted for different formatting purposes.

For illustration, the following is a sample name object.

```
person_name ⇒
   prefix  ⇒ "President"
   given ⇒ "John"
   middle ⇒ "Fitzgerald"
   surname ⇒ "Kennedy"
   nameLocale ⇒ "und-US" // this is just for illustration
```

```
nameOrder ⇒ “givenFirst” // this too
```

A name object is logically composed of the fields above (and others). There must be at least one field present: either a `given` or `surname` field. Other fields are optional, and some of them can be constructed from other fields if necessary.

Note an application could have (or generate) name data with additional fields for monogram, monogram-narrow or surname prefix (tuessenvoegsels), rather than using the formatting described below. In such cases, it is presumed that the implementation will use those values instead of formatting the name with CLDR data.

So if an application supports a choice flag, allowing either “John” or “Jack” as the preferred given name, then the application would construct the appropriate name object to be formatted, based on the flag. The format where both names are present, such as ‘John “Jack” Kennedy’, is rarely used and not considered within the scope of this document.

For detailed examples, see 5.1 Example Name field elements.

Person Name Attributes

A person name may have any of three attributes: length, style, and usage.

Each of these attributes are described below using the sample name object above as an example.

length

The `length` attribute specifies the relative length of a formatted name depending on context. For example, a `long` formal name in English would include prefix, given, middle, surname plus suffix; whereas `short` may only be the given name.

Note that the formats may be the same for different lengths depending on the style, usage, and cultural conventions for the locale. For example, medium and short may be the same for a particular context.

Note that we considered a “full” length attribute but could not find a justifiable use case for that.

Parameter	**Description**
`long`	A `long` length would usually include all parts needed for a legal name or identification. usage=”referring”, style=”formal” *“John Fitzgerald Kennedy”*

`medium`	A `medium` length is between long and short usage=”referring”, style=”formal” *“John Kennedy”*
`short`	A `short` length uses a minimum set of names. usage=”referring”, style=”formal” *“John”*
`monogram`	The `monogram` length is very short, usually initials (eg, JS, EvP) usage=”referring”, style=”formal” *“JFK”*
`monogram-narrow`	The `monogram-narrow` length would usually be a single grapheme cluster (eg, J); typically the first grapheme cluster in the given name or surname usage=”referring”, style=”formal” *“K”*

style

The style indicates the formality of usage. A name on a badge for an informal club gathering may be much different from an award announcement at the Nobel Prize Ceremonies.

Note that the formats may be the same for different styles depending on the length, usage, and cultural conventions for the locale. For example short formal and short informal may both be just the given name.

Parameter	**Description**
`formal`	A more formal name for the individual. The composition depends upon the language. For example, a language might include the prefix and suffix and a full middle name in the long form. length=”medium”, style=”formal” “John F. Kennedy”
`informal`	A less formal name for the individual. The composition depends upon the language. For example, a language might exclude the prefix, suffix and middle name. Depending on the length, it may also exclude the surname. length=”medium”, style=”informal” “John Kennedy”

usage

The usage indicates if the formatted name is being used to address someone, refer to someone, or sort their name with other names.

Again, the format may be the same in some cases for some cultures where usage is different. For example: `usage=sorting` may have the same format as `usage=addressing`, if the language uses surname first.

Parameter	**Description**
`addressing`	Used when speaking "to" a person example: "John" [medium, informal]
`referring`	Used when speaking "about" a person, or "nominative" case example: "John Kennedy" [medium, informal]
`sorting`	Used for a sorted list example: "Kennedy, John" [medium, informal]

Russian provides a good example of `addressing` vs `referring`. An example *ru-Cyrl* name object:

```
person_name ⇒
   prefix  ⇒ "г-н"    // "Mr"
   given   ⇒ "Иван"   // "Ivan"
   middle  ⇒ "Петрович"     // "Petrovich"
   surname      ⇒ "Васильев"     // "Vasiliev"
```

In Russian, when *`addressing`* a person (with `length=long`), it might be

- г-н Иван Петрович Васильев   `// "Mr Ivan Petrovich Vasiliev"`

And when *`referring`* to a person, it might be

- Васильев Иван Петрович `// "Vasiliev Ivan Petrovich"`

order

The order indicates where a locale has two different orders for patterns, based on the locale of the name object.

Again, the format may be the same in some cases for some locales . For example, if the order is always "given before surname" (GBS) for the locale, then the order attribute can always be omitted.

Parameter	**Description**

`surnameFirst`	The surname precedes the given name. Abbreviated as SBG
`givenFirst`	The given name precedes the surname. Abbreviated as GBS

For example, when the display language is Japanese, it is customary to use SBG for names of people from Japan and Hungary, but use GBS for names of people from the United States and France.

Name Order

The name order data provides a mapping from the name object locale to an order parameter. In current data, the important part of the name object locale is the region code, but allowing a locale code provides for easy extension in the future.

```
<!ELEMENT nameOrder EMPTY >
```

```
<!ATTLIST nameOrder nameLocales NMTOKENS #REQUIRED >
<!--@MATCH:validity/set/locale
```

- The NMTOKENS are a space delimited list of unicode_locale_ids. Normally they are limited to language, script, and region.

```
<!ATTLIST nameOrder order (surnameFirst | givenFirst) #REQUIRED >
```

The nameOrder values are used with a name object's fields to derive an order, as follows:

1. If the name object has a nameOrder field, then return that field's value
2. Otherwise, if the name object has a locale field, and there is a nameOrder element:
    a. Map that locale field using the likelySubtags data, eg zh ⇒ zh-Hans-CN
    b. Traverse the nameOrder elements, and within each order attribute value to see if there is a match.
    c. As usual, a **und** (undefined) language subtag or empty script or region in the order attribute locale matches any language subtag, script subtag, or region subtag (respectively) in the name object's locale field.
    d. If there is a match, return the nameOrder elements order attribute value.
3. Otherwise, return givenFirst

For example, the data for Japanese might look like the following:

<nameOrder

nameLocales=”zh ja und-CN und-TW und-SG und-HK und-MO und-HU und-JP”

order=”surnameFirst”>

The nameLocales will match any locale with zh unicode_language_subtag or any locale with a CN, TW, SG, HK MO, HU, or JP unicode_region_subtag.

Here are some examples. Note that if there is no order field or locale field in the name object, then the result is givenFirst.

Name Object order	**Name Object locale**	**Resulting Order**
surnameFirst		surnameFirst
	zh	surnameFirst
	und-JP	surnameFirst
	fr	givenFirst
		givenFirst

Open Issue

Languages such as Japanese may require different *personName* format data to handle the different case of romaji names in Japan with spaces between elements (e.g. “*KANŌ Jigorō*”), Kana names with an *interpunct*, ‘・’ U+30FB katakana middle dot between elements (e.g. “かのう・じごろう”), and Kanji names with no spaces (“加藤澤男”). We are considering a mechanism like nameOrder to allow for these choices.

3. namePattern Syntax

A *namePattern* is composed of a sequence of field IDs, each enclosed in curly braces, and separated by zero or more literal characters (eg, space or comma + space). An Extended Backus Normal Form (EBNF) is used to describe the namePattern format for a specific set of attributes. It has the following structure. This is the `( #PCDATA )` reference in the element specification above.

	EBNF	**Comments**
namePattern	= literal ('{' modField '}' literal)+ ;	

modField	= field (’-’ modifier)*;	Each modifier may only be used once; modifiers must be in alphabetical order.
field	= 'prefix' | 'given' | 'middle' | 'surname' | 'surname2' | 'suffix' ;	Parts of a name, described below
modifier	= 'allcaps' | 'initial' ;	Optional modifiers that can be applied to name parts.
literal	= codepoint* ;	Zero or more Unicode codepoints.

This uses a flat structure (like some other structures in CLDR), representing the fact that the full array is sparse at different levels. The flat structure allows for a more tuned fallback.

Fields

The proposal assumes that the name data to be formatted consists of the fields in the table below. The examples in the table may refer to part of the full name “*President George Herbert Walker van Gogh, PhD*”. All of the fields may contain multiple words. Field IDs are lowercase ASCII alphanumeric, and start with an alphabetic character.

When determining how a full name is to be placed into name fields, the data should be organized functionally. That is, if a name part is on the dividing line between `middle` and `given`, the key feature is whether it would always occur with the rest of the given name. For example, in “*Mary Jean Smith*”, if *Mary* never occurs without the “*Jean”*, then the given name should be “*Mary Jean*”. If “*Smith*” never occurs without the “*Jean*”, the `surname` should be “*Jean Smith*”. Otherwise, “*Jean*” would be the `middle` name.

For example, a patronymic would be treated as a `middle` name in most slavic languages.

In some cultures, two surnames are used to indicate the paternal and maternal family names or generational names indicating father, grandfather. The `surname2` field is used to indicate this. This specification assumes that if a name object does not have two surnames, then the surname2 field is not populated. (That is, no pattern should have a surname2 field without a surname field.) Order of fields in a pattern can vary arbitrarily by locale.

In most cultures, there is a concept of nickname or preferred name, which is used in informal settings or sometimes as a “public” or “stage name”. For purposes of the CLDR PersonName formats, these alternate names should be presented for formatting as separate name objects.

Field	**Description**
`prefix`	Typically a title, honorific, or generational qualifier. Example: ‘Ms.’, ‘Mr.’, ’Dr’, ‘President’
`given`	The “given” name. Can be multiple words such as “Mary Ann”. Examples: “George”, “Mary Jean”, or “Jean-Louis”
`middle`	A middle name, usually names(s) written between the given and surname. Can be multiple words. In some references, also known as a “second” or “additional” given name. Examples: “Herbert Walker” as in `{ given: “George”, `**`middle: “Herbert Walker”`**`, surname: “Bush” }` “S.” as in “Harry S. Truman”. Yes, his middle name was legally “S.”.
`surname`	The “family name”. Can be more than one word. Example: “van Gogh” as in `{ given: “Vincent”, middle: “Willem”, `**`surname: “van Gogh”`**` }` Other examples: “Heathcote-Drummond-Willoughby” as in “William Emanuel Heathcote-Drummond-Willoughby III”

`surname2`	Secondary surname (used in some cultures), such as second or maternal surname in Mexico and Spain.
	
	Example: “Barrientos” in “Diego Rivera Barrientos”;
	
	`{ given: “Diego”, surname: “Rivera”, `**`surname2: “Barrientos”`**` }`
	
	For name with given middle, surname, surname2, see https://en.wikipedia.org/wiki/Spanish_naming_customs#Foreign_citizens - "Mary Jane Smith" moves to Spain and new name may be
	
	`{ given: "Mary", middle: "Jane", surname: "Smith", surname2: "Jones" }`

`suffix`	Typically a title, honorific, or generational qualifier Example: “PhD”, “Jr.” Example: “Sammy Davis Jr.” (link) `{ given: “Samuel”, middle: “George”, surname: “Davis”, `**`suffix: “Jr.”`**` }`

	an alternate name object may be presented for formatting using the “stage” name from the application’s data: `{ given: “Sammy”, middle: “”, surname: “Davis”,` **`suffix: “Jr.”`** `}`

The choice of whether to put a particular part of somebody’s name into the give-name or middle-name field depends on how the name is used — for the most part, the middle-name field is used for parts of the name that can be omitted when the formatted name is shortened for presentation. If two words are always kept together, they should both go into the same field.

For example, in “Mary Jane Smith”, if the person goes by “Mary Jane,” both names would go into the `given` name field, but if she goes by “Mary” and drops “Jane” in less formal contexts, then “Jane” would go into the `middle` name field. Similar things would happen with the surnames.

Some other examples:

- British name: *John Ronald Reuel Tolkien*: `given` name is "John", `middle` name would be "Ronald Reuel", and the `surame` is "Tolkien".
- Dutch name: *Anneliese Louise van der Pol*: `given` name: "Anneliese", `middle` name: "Louise", `surname`: "van der Pol"
- French name: “Jean-Louis Trintignant” would not be Jean (given) Louis (middle) Trintignant (surname), since “Louis” wouldn’t be discarded when formatting. Instead it would be Jean-Louis (given) Trintignant (surname)

Note: If the legal name, stage name, etc. are substantially different, then that information can be logically in a separate Name record. That is, it is up to the implementation to maintain any distinctions that are important to it: this specification is focusing on formatting a Name record that is given to it.

`surname2` would only be asked for in certain locales, and where it is “splittable” or considered a separate divisible name, such as in Mexico or Spain. For instance, in Mexico, the first and second surname are used for the legal name and in formal settings, and sometimes only the first surname is used in familiar or informal contexts.

- Heathcote-Drummond is a single surname and wouldn’t be {surname}-{surname2} because we would never discard part of the name when formatting.
- Spanish name: "Jose Luis Garcia Barrientos":   The `given` name is “Jose’, the `middle` name is “Luis”, the `surname` is "Garcia”, and the `surname2` is “Barrientos"

How the names get placed into the fields is beyond the scope of this specification; this document just lays out the assumptions the formatting code makes when formatting the names.

Open Issue: gender-specific formatting

In Arabic {given="Farah", middle="Fadi", surname="Sajid Al-Tikriti"} is formatted as "Farah <u>bint</u> Fadi <u>bin</u> Sajid Al-Tikriti". However, the "bint" can change to "bin" depending on the gender of the name. The design doesn't yet allow this, except for having the name object insert the "bint" or "bin".

Modifiers

Each field in a pattern can have one or more modifiers. These are currently `initial` and `allcaps`, which indicate the initial grapheme or all capital letters respectively. The modifiers can be appended to any field name, such as `{given-initial}` for the first letter of the givenname. If more than one modifier is applied, they should appear in alphabetical order, and each modifier may occur at most once. The modifiers transform the input data in the following way:

Modifier	**Description**
allcaps	In some contexts, it is desired to represent an element in all caps. For example, a new guideline in Japan is that for the Latin representation of Japanese names, the family name comes first and is presented in all capitals. This would be represented as "{surname-allcaps} {given}" So, Hayao Miyazaki (宮崎 駿) would be represented in Latin characters in Japan (ja-Latn-JP) as "*MIYAZAKI Hayao*" *The default implementation uses the default Unicode uppercase algorithm; if the name object has a locale, and CLDR supports a locale-specific algorithm for that locale, then that algorithm is used. The name object can override this, as detailed below.*
initial	Example a name such as { given: "Lyndon", middle: "Baines", surname: "Johnson" } could be represented as "{given-initial}{middle-initial}{surname-initial}" or "**LBJ**" *The default implementation uses the first grapheme cluster of the value for the field; if the name object has a locale, and CLDR supports a locale-specific grapheme cluster algorithm for that locale, then that algorithm is used. The name object can override this, as detailed below.*

There may be more modifiers in the future.

Open Issue: modifier details

To make the exposition simpler, we might move the following into an annex.

An implementation may choose to support *enhanced* modifiers via the name object.

For example, if you had a surname of "***de Souza***", "`{surname-initial}`" would produce "***d***" by default and "`{surname-allcaps-initial}`" would produce "***D***" by default.

However, 'd' and 'D' are typically incorrect values, because such particles are typically skipped in producing initials: the desired value would be 'S'.

We address this by allowing the calling code to supply additional information in the name object — the calling code can (logically) add a "surname-initial" field that specifies the letter to use as the surname initial. The name formatter will look for that first and, if it's not present, fall back on just taking the first grapheme cluster.

So in this example, when the implementation is queried for `{surname-initial}`, it can skip over the "de ". That is, for now it is the responsibility of the name object to return a "surname-initial" field with value "S" to give the proper result.

Here is how this works. The name formatter queries the name object for each possible combination of the specified modifiers, and if any one of the combinations produces a result, then that result is used (with any remaining modifiers applied algorithmically).

The order of the queried sets of modifiers is:

1. longest list first
2. among lists of the sample length, in alphabetical order of modifiers

Example1:

With `{surname-allcaps-initial}`, the formatter will ask the name object for items in the following order:

```
1. surname-allcaps-initial
2. surname-allcaps
3. surname-initial
4. surname
```

If the name object has `surname-allcaps` (but not `surname-allcaps-initial`) then the unused `initial` modifier is applied algorithmically to the result.

Example 2:

With a field F and three modifiers A-B-C the order is the following.

F-A-B-C; F-A-B, F-A-C, F-B-C; F-A, F-B, F-C; F

As above, remaining modifiers are algorithmically applied. So if the first result is obtained by F-B, then A and then C are algorithmically applied to the result.

Future versions of CLDR may supply rules and data for handling locale-specific initials, including the above examples of particles / tussenvoegsel, and the following examples.

Example 1.

Greek initials can be produced via the following process in the name object, and returned to the formatter.

- Include all letters up through the first consonant or digraph (including the consonant or digraph). (This is a simplified version of the actual process.)

Examples:

- Χριστίνα Λόπεζ (Christina Lopez) ⟶ Χ. Λόπεζ (C. Lopez)
- Ντέιβιντ Λόπεζ (David Lopez) ⟶ Ντ. Λόπεζ (D. Lopez)

  Note that Ντ is a digraph representing the sound D.

Example 2. The name object could supply information based on an implementation's information about the person's name in different scripts. When formatting Kennedy's name "john fitzgerald kennedy" (ジョン・フィッツジェラルド・ケネディ) in display_locale=ja, when asked for {middle-initial}, it could look up the original English version of the name (if available) and supply "ジョン・F・ケネディ" — note the a Latin-script "F" in that result. Similarly, for display_locale=ko, the implementation can produce the desired result is "존 F. 케네디" with punctuation appropriate to Korean.

Example 3.

To make an initial when there are multiple words, an implementation might produce the following:

- George H. W. Bush ⇒ {middle-initial} producing "H.W.".

- Erik Martin van der Poel: {middle-initial} producing “V” by default, but might produce “vdP” or P in other languages.
- A field containing multiple words might not actually initialize all of them, such as in “Mohammed bin Ali bin Osman” (“MAO”).
- John Ronald Reuel Tolkien as “J.R.R. Tolkien” from { given: “John”, middle: “Ronald Reuel”, surname: “Tolkien” }
- The short version of "Son Heung-min" is "H. Son" and not "H. M. Son" or the like. Korean given-names have hyphens and the part after the hyphen is lower-case.

Many cultures and contexts may expect initials for all words in a field, but it needs to be determined what character(s) may be used as a delimiter between words in a field, and how to indicate if it is preferred to use the initial of only word, or all words. Often, this requires language-specific knowledge that is beyond the current scope of this document.

4. Person Name Formatting

The patterns are in personName elements, which are themselves in a personNames element.
The minimum set of name patterns must include the following:

```
<personNames>
  <personName length="long" usage="referring addressing" style="informal">...</personName>
  <personName length="medium" usage="referring" style="formal"> ...</personName>
  <personName length="medium" usage="referring" style="informal">...</personName>
  <personName length="short" usage="referring" style="informal">...</personName>
  <personName >...</personName>
</personNames>
```

The last personName element must have no attributes for the names data to be well-formed; that is the ultimate fallback formatting element.

4.1. Choosing a personName

The personName metadata in CLDR provides representations for how names are to be formatted across the different axes of *length*, *usage*, and *style*. More than one `namePattern` can be associated with a single `personName` entry. An algorithm is then used to choose the best `namePattern` to use.

As an example for English, this may look like:

```
<personNames>
    <personName length="long" usage="referring" style="formal">
       <namePattern>{prefix} {given} {middle} {surname}, {suffix}</namePattern>
    </personName>
    <personName length="long" usage="referring" style="informal">
      <namePattern>{given} «{middle}» {surname}</namePattern>
      <namePattern>«{middle}» {surname}</namePattern>
    </personName>
    <personName length="long" usage="sorting" style="informal">
      <namePattern>{surname}, {given} {middle}</namePattern>
    </personName>
...
</personNames>
```

The task is to find the best personName for a given set of input attributes. Well-formed data will always cover all possible combinations of the input parameters, so the algorithm is simple: traverse the list of person names until the first match is found, then return it.

In more detail:

A set of input parameters {length=L usage=U style=S} matches a personName element when:

- The length attribute values contain L or there is no length attribute, and
- The usage attribute values contain U or there is no usage attribute, and
- The style attribute values contain S or there is no style attribute

Example for input parameters

length=**long** usage=**referring** style=**formal**

To match a personName, all three attributes in the personName must match:

Sample personName attributes	**Matches?**	**Comment**
`length="long" usage="referring" style="formal"`	Y	exact match
`length="long" usage="referring" style="informal"`	N	mismatch for style
`length="long" usage="addressing referring" style="formal"`	Y	param ∈ usage
`length="long" style="formal"`	Y	missing usage = all!

To find the matching personName element, traverse all the personNames in order until the first one is found. This will always terminate since the data is well-formed in CLDR. (The algorithm can be modified to handle ill-formed data by taking the last personName if there is otherwise no match.)

Implementation Guidelines

The format of the personName elements allows for a condensed version of the personName data. It can be easily expanded into a set with the exact personName attributes, where order does not matter. It would contain one line for each of the combinations of single attributes, thus 5 x 2 x 3 x 2 or 60 separate lines in the current version. (That number could grow in the future.)

```
<personName length="long" style="formal" usage="sorting" nameOrder=”surnameFirst”/>
<personName length="long" style="formal" usage="referring" nameOrder=”surnameFirst”/>
...
```

The expansion can be done by implementations that favor speed over memory consumption, for example. It also may be done when presenting data to translators, since the sets of attributes that have the same patterns will be different in different languages.

Conversely, the format with 60 separate lines can be mechanically condensed down to a smaller number of lines, such as about a dozen for English. Such a “condensation” is only valid if evaluating it produces exactly the same results as the fully expanded version would.

Implementations that gather data, such as the CLDR Survey Tool, can also perform tests of data consistency, such as where the order of fields unexpectedly differs: *surname* before *given* in one line for usage=sorting, but *given* before *surname* in another.

4.2. Choosing a namePattern

To format a name, the fields in a namePattern are replaced with the actual name element values from the input record. The personName element can contain multiple namePattern elements. We choose one based on the fields in the name record that are populated: The namePattern that has the fewest unpopulated fields wins (if more than one have the same number of unpopulated fields, the first one wins).
If the “winning” namePattern still has fields that are unpopulated in the name record, we alter the pattern algorithmically as follows:

1. For each *empty* field from the start of the pattern, that field and all whitespace and literal text between it and the next field is deleted. (This process stops with the first non-empty field.)
2. For each empty field from the end of the pattern, that field and all whitespace and literal text between it and the previous field is deleted. (This process stops with the first non-empty field.)
3. For each empty field in the middle of the pattern (going from left to right), that field and all literal text between it and the nearest whitespace or field on both sides is deleted. If this results in two whitespace characters next to each other, they are coalesced into one.

4.2.1 Examples of choosing a namePattern

Examples for rules 1 and 2:

The personName element contains:

<namePattern>{prefix} {given} {middle} {surname}, {suffix}</namePattern>

The input name record contains:

prefix	given	middle	surname	suffix
	Raymond	J.	Johnson	Jr.

The output is:

Raymond J. Johnson, Jr.

The “prefix” field is empty, and so both it and the space that follows it are omitted from the output, according to rule 1 above.

If, instead, the input name record contains:

prefix	given	middle	surname	suffix
	Raymond	J.	Johnson	

The output is:

Raymond J. Johnson

The “prefix” field is empty, and so both it and the space that follows it are omitted from the output, according to rule 1 above.

The “suffix” field is also empty, so it and both the comma and the space that precede it are omitted from the output, according to rule 2 above.

Examples for rule 3 and the interaction between the rules:

To see how rule 3 interacts with the other rules, consider an imaginary language in which people generally have given and middle names, and the middle name is always written with parentheses around it, and the given name is usually written as an initial with a following period.

The personName element contains:

<namePattern>{given-initial}. ({middle}) {surname}</namePattern>

The input name record contains:

given	middle	surname
Foo	Bar	Baz

The output is:

F. (Bar) Baz

If, instead, the input name record contains:

given	middle	surname
Foo		Baz

The output is:

F. Baz

The “middle” field is empty, so it and the surrounding parentheses are omitted from the output, as is one of the surrounding spaces, according to rule 3. The period after “{given-initial}” remains, because it is separated from the “{middle}” element by space-- punctuation around a missing field is only deleted up until the closest space in each direction.

If there were no space between the period and the parentheses, as might happen if our hypothetical language didn’t use spaces:

```
<namePattern>{given-initial}.({middle}) {surname}</namePattern>
```

The input name record still contains:

given	middle	surname
Foo		Baz

The output is:

F Baz

Both the period after “{given-initial}” *and* the parentheses around “{middle}” are omitted from the output, because there was no space between them-- instead, we delete punctuation all the way up to the neighboring field. To solve this (making sure the “{given-initial}” field always has a period after it), you would add another namePattern:

```
<namePattern>{given-initial}.({middle}) {surname}</namePattern>
<namePattern>{given-initial}. {surname}</namePattern>
```

The first pattern would be used when the “middle” field is populated, and the second pattern would be used when the “middle” field is empty.

Rules 1 and 3 can conflict in similar ways. If the personName element contains (there’s a space between the period and the opening parenthesis again):

```
<namePattern>{given-initial}. ({middle}) {surname}</namePattern>
```

And the input name record contains:

given	middle	surname
	Bar	Baz

The output is:

Bar) Baz

Because the “given” field is empty, rule 1 not only has us delete it, but also all punctuation up to “{middle}”. This includes *both* the period *and* the opening parenthesis. Again, to solve this, you’d supply two namePatterns:

```
<namePattern>{given-initial}. ({middle}) {surname}</namePattern>
<namePattern> ({middle}) {surname}</namePattern>
```

The output would then be:

(Bar) Baz

The first namePattern would be used if the “given” field was populated, and the second would be used if it was empty.

If, instead, the input name record contains:

given	middle	surname
Foo		Baz

The output is:

F. Baz

5. Examples

5.1 Example name objects

locale	prefix	given	middle	surname	surname2	suffix	link
en-US	Lt.	John	Fitzgerald	Kennedy		Jr.	wikipedia
en-US	.	Jack		Kennedy			wikipedia
es-EC	General	Luis	Telmo	Paz y Miño	Estrella		wikipedia
es-MX		Diego		Rivera	Barrientos		
id-ID	Pak	Sukarno					wikipedia
id-ID	Pak	Mahyadi		Panggabean			wikipedia
ar	الشيخ	محمد	الفاضل	بن محمد الطاهر	بن عاشور		wikipedia.ar
ar-Latn	Sheikh	Muḥammad	Fadhel	Muhammad al-Tahir	Ben Achour		wikipedia
ar-SA		مطلق	بن حميد بن أحمد	الثبيتي	العتيبي		wikipewdia.ar
ar-Latn-SA		Mutlaq	bin Humaid bin Ahmed	Al-Thubeiti	Al-Otaibi		wikipedia
ru-RU	г-н	Александр	Исаевич	Солженицын			wikipedia
ja-JP		嘉納		治五郎		教授	
ja-Hrkt		じごろう		かのう			wikipedia.ja

ja-Latn	Prof	Jigorō		Kanō			wikipedia
th-TH	นาย	วรายุทธ		เย็นบำรุง			wikipedia.th
th-Latn	Mr	Varayuth		Yenbamroong			wikipedia
nl-NL	Ms	Anneliese	Louise	van der Pol			wikipedia
nl-NL	Mevr.	Anneliese	Louise	Pol	van der		wikipedia

Ibn Arabi: “**Abū ʿAbd Allāh Muḥammad ibn ʿAlī ibn Muḥammad ibn al-ʿArabī al-Ḥātimī al-Ṭāʾī al-Andalusī al-Mursī al-Dimashqī**”

5.2 Example personName formats

The following provides a list of sample formats for some locales.

As a reminder:

- length values are {*long, medium, short, monogram, monogram-narrow*}
- style values are {*formal, informal*}
- usage values are {*addressing, referring, sorting*}
- field value are {`{prefix}, {given}, {middle}, {surname}, {surname2}, {suffix}`} (with optional modifier suffixes)

Note that these formats provide a full matrix of length × style × usage, but they could be collapsed by sharing formats using multiple attribute values, and ordering them the right way. In localization, we’d expand into the full matrix for translators, and collapse them back in the production data. An implementation may choose to use either the collapsed form or an expanded form, based on memory and performance considerations.

locale: en-US

```
<personNames>
<!-- monograms. Note that missing style, usage mean all match-->
 <personName length="monogram-narrow">
   <namePattern>{given-initial}
   </namePattern>
 </personName>
 <personName length="monogram">
   <namePattern>{given-initial}{surname-initial}
   </namePattern>
 </personName>
<!-- short -->
<!-- In English short, formal = surname (example), in sorting, referring & addressing -->
```

```
 <personName length="short" style="formal">
   <namePattern>{prefix} {surname}
   </namePattern>
 </personName>
<!-- Because formal is already handled, no usage = referring addressing -->
 <personName length="short">
   <namePattern>{given}
   </namePattern>
 </personName>
<!-- medium -->
 <personName length="medium" usage="sorting">
   <namePattern>{surname}, {given}
   </namePattern>
 </personName>
<!-- Formal referring e.g. “Jonas Salk MD”. This is for illustration, since English doesn’t
tend to have as strong a difference as some other languages. -->
 <personName length="medium" style="formal" usage="referring">
   <namePattern>{given} {surname} {suffix}
   </namePattern>
 </personName>
<!-- Because referring is already handled, formal addressing “Dr Jonas Salk” -->
 <personName length="medium" style="formal">
   <namePattern>{prefix} {given} {surname}
   </namePattern>
 </personName>
<!-- Informal - all usage cases, addressing and referring -->
 <personName length="medium" style="informal">
   <namePattern>{given} {surname}
   </namePattern>
 </personName>
<!-- long -->
 <personName length="long" usage="sorting">
   <namePattern>{surname}, {prefix} {given} {middle} {suffix}
   </namePattern>
 </personName>
 <personName length="long" style="formal">
   <namePattern>{prefix} {given} {middle} {surname} {suffix}
   </namePattern>
 </personName>
<!-- Ultimate fallback to handle all remaining cases -->
 <personName>
   <namePattern>{prefix} {given} {middle} {surname} {suffix}
   </namePattern>
 </personName>
<personNames>
```

locale: es-MX

Spanish for Mexico is chosen as an example to illustrate the use of the surname2 field. Virtually everyone in Mexico and Spain have two surnames, usually the paternal surname and maternal surname, however recent reforms in Mexico now have government forms requesting the first surname (*primer apellido*) and second surname (*segundo apellido*).

```
<personNames>
   <!-- monograms. Note that missing style, usage mean all match-->
   <personName length="monogram-narrow" style="formal">
       <namePattern>{surname-initial}
       </namePattern>
   </personName>
   <personName length="monogram-narrow">
       <namePattern>{given-initial}
       </namePattern>
   </personName>
   <personName length="monogram" style="formal">
       <!-- In Spanish monogram, use formal with surname2 -->
       <namePattern>{given-initial}{surname-initial}{surname2-initial}
       </namePattern>
   </personName>
   <personName length="monogram">
       <!-- In Spanish monogram, if not formal, then informal -->
       <namePattern>{given-initial}{surname-initial}
       </namePattern>
   </personName>
   <!-- short -->
   <!-- In Spanish short, formal = surname surname2 (example), in sorting, referring &
addressing -->
   <personName length="short" style="formal" usage="addressing">
       <namePattern>{prefix} {surname} {surname2}
       </namePattern>
   </personName>
   <!-- Because addressing is already handled, assume referring -->
   <personName length="short" style="formal">
       <namePattern>{surname} {surname2}
       </namePattern>
   </personName>
   <!-- Because formal is already handled, no usage = referring addressing -->
   <!-- To get the commonly used {surname} only case, used in informal "additional" reference
in articles,
     == the application would need to manually remove the contents of {surname2}
     == do we need an additional usage case?
       <personName length="short">
           <namePattern>{surname}
           </namePattern>
       </personName>
   -->
   <!-- Because formal is already handled, "formal" will assume the {given} name"
       <personName length="short" style = "formal">
           <namePattern>{given}
           </namePattern>
       </personName>
   -->
   <!-- Because formal is already handled, or if {given} not found, assume addressing -->
   <personName length="short">
       <namePattern>{given}
       </namePattern>
   </personName>
   <!-- medium -->
   <personName length="medium" usage="sorting">
       <namePattern>{surname}, {given} {middle}
       </namePattern>
   </personName>
   <personName length="medium" style="formal">
       <namePattern>{prefix} {given} {surname} {surname2} {suffix}
       </namePattern>
   </personName>
   <personName length="medium" style="informal">
       <namePattern>{given} {surname}
       </namePattern>
   </personName>
   <!-- long -->
   <!-- good reference:
https://www.councilscienceeditors.org/wp-content/uploads/v26n4p118-121.pdf -->
```

```
   <personName length="long" usage="sorting">
       <namePattern>{surname} {surname2}, {prefix} {given} {middle} {suffix}
       </namePattern>
   </personName>
   <personName length="long" style="formal">
       <namePattern>{prefix} {given} {middle} {surname} {surname2} {suffix}
       </namePattern>
   </personName>
   <!-- Ultimate fallback to handle all remaining cases -->
   <personName>
       <namePattern>{prefix} {given} {middle} {surname} {surname2} {suffix}
       </namePattern>
   </personName>
<personNames>
```

locale: ja-JP

Japanese is chosen to show surname first and formatting with no spaces.

```
<personNames>
   <!-- ja-JP. Formatting names in Japanese, script: Jpan -->
   <!-- monograms. There is no "initial" in Japanese, use the family name -->
   <personName length="monogram-narrow monogram">
       <namePattern>{surname}
       </namePattern>
   </personName>
   <!-- short -->
   <personName length="short" style="formal">
       <namePattern>{surname}
       </namePattern>
   </personName>
   <!-- Because formal is already handled, no usage = referring addressing -->
   <personName length="short" style="informal">
       <namePattern>{given}
       </namePattern>
   </personName>
```

```
    <!-- medium -->
    <!-- Note that Japanese names usually sort by their phonetic or "kana" values -->
    <personName length="medium" usage="sorting">
        <namePattern>{surname}{given}
        </namePattern>
    </personName>
    <personName length="medium" style="formal">
        <namePattern>{prefix}{surname}{given}{suffix}
        </namePattern>
    </personName>
    <personName length="medium" style="informal">
        <namePattern>{surname}{given}
        </namePattern>
    </personName>
    <!-- long -->
    <personName length="long" usage="sorting">
        <!-- Middle name is not generally used in Japanese -->
       <!-- included here in case middle is populated -->
        <namePattern>{surname}{given}{middle}{prefix}{suffix}
        </namePattern>
    </personName>
    <personName length="long" style="formal">
        <namePattern>{prefix}{surname}{given}{middle}{suffix}
        </namePattern>
    </personName>
    <!-- Ultimate fallback to handle all remaining cases -->
    <personName>
        <namePattern>{prefix}{surname}{given}{middle}{suffix}
        </namePattern>
    </personName>
<personNames>
```

locale ja-Latn-JP

This example shows the special case required by the Japanese government for presenting Japanese names in Latin as surname first with the surname in all caps. Middle names are included in the length=”long” *personName* formats even though Japanese do not usually have middle names, to allow for possibly populated fields for non-Japanese names that may be formatted in the JP region. If the name object has an empty middle name, it will be trimmed out by the formatting algorithm.

```
<personNames>
   <!-- ja-Latn-JP. Formatting names in Japanese Romaji, script: Latn -->
   <!-- monograms. There is no "initial" in Japanese, use the family name -->
   <personName length="monogram-narrow">
       <namePattern>{surname-initial}
       </namePattern>
   </personName>
   <personName length="monogram">
       <namePattern>{surname-initial}{given-initial}
       </namePattern>
   </personName>
   <!-- short -->
   <personName length="short" style="formal">
       <namePattern>{surname}
       </namePattern>
   </personName>
   <!-- Because formal is already handled, no usage = referring addressing -->
   <personName length="short" style="informal">
       <namePattern>{given}
       </namePattern>
   </personName>
   <!-- medium -->
   <!-- Note that Japanese names usually sort by their phonetic or "kana" values -->
   <personName length="medium" usage="sorting">
       <namePattern>{surname-allcaps} {given}
       </namePattern>
   </personName>
   <personName length="medium" style="formal">
       <namePattern>{prefix} {surname-allcaps} {given} {suffix}
       </namePattern>
   </personName>
   <personName length="medium" style="informal">
       <namePattern>{surname-allcaps} {given}
       </namePattern>
   </personName>
   <!-- long -->
   <personName length="long" usage="sorting">
       <!-- Middle name is not generally used in Japanese -->
       <namePattern>{surname-allcaps} {given} {middle} {prefix} {suffix}
       </namePattern>
   </personName>
   <personName length="long" style="formal">
       <namePattern>{prefix} {surname-allcaps} {given} {middle} {suffix}
       </namePattern>
   </personName>
   <!-- Ultimate fallback to handle all remaining cases -->
   <personName>
       <namePattern>{prefix} {surname-allcaps} {given} {middle} {suffix}
       </namePattern>
       <namePattern>{prefix} {surname-allcaps} {given} {middle} {suffix}
       </namePattern>
   </personName>
<personNames>
```

6. Sample API

Here's an example of what the API of a name formatter that uses this new CLDR data might look like in Java. This is far from the only way to structure an appropriate API, and an API in some other programming language (such as C++) would necessarily have a different structure.

```
public enum Field {
    PREFIX, GIVEN, MIDDLE, SURNAME, SURNAME2, SUFFIX
};
public enum Modifier { INITIAL, ALL_CAPS };

public interface PersonNameFields {
     public String getField(Field base, Modifier modifier);
};

public class PersonNameFormatter {
     public enum Length {
          LONG,
          MEDIUM,
          SHORT,
          MONOGRAM,
          MONOGRAM_NARROW
     };

     public enum Style {
          FORMAL,
          INFORMAL
     };

     public enum Usage {
          REFERRING,
          ADDRESSING,
          SORTING
     };

     public PersonNameFormatter(Locale locale, Length length, Style
style, Usage usage);

     public String format(PersonNameFields nameFields);
};
```

Note that in this example, we show the `PersonNameFields` object as a Java `interface` that has one getter that takes two enum values to specify which value to get. A simple implementation could, of course, just use a simple struct with all of the named fields.
What we're trying to show here instead is that the `PersonNameFields` is intended not necessarily to contain the field values themselves, but to act as an intermediary between the `PersonNameFormatter` and the client application's *actual* repository of name data. Having one getter that effectively takes a field ID gives the client code more flexibility in implementation: If, for example each field is fetched with a SQL query and the only thing in the query that changes for different fields is the field ID, having one getter with a field-ID parameter allows for a simpler implementation.

If the implementation also allows for client applications to override the default implementations of the initials and allcaps modifiers, this API design saves the implementer from having to supply `xxx`, `xxx-initial`, `xxx-allcaps`, and `xxx-allcaps-initial` fields for all of the variations of each field (compared to using a plain struct, where you'd just have to have separate fields for all of the variations), and allows the implementing class to calculate the modified field values on the fly as needed.

7. References

LDML TR#35 - Locale Data Markup Language

Standards

- LDAP – Lightweight Directory Access Protocol + RFC4519
  https://www.iana.org/assignments/ldap-parameters/ldap-parameters.xhtml
  https://docs.ldap.com/specs/rfc4519.txt
- ITU X.20
  https://www.itu.int/itu-t/recommendations/rec.aspx?rec=X.520
- hCard
  http://microformats.org/wiki/hcard
- W3C HTML autofill fields
  https://www.w3.org/TR/html52/sec-forms.html#autofill-field
- OASIS xNL
  http://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.html#_Toc213384944
- UPU S42, ADIS, ISO 19160
  http://xml.coverpages.org/ISO-FocusPlus-AddressingStandards-2010-06.pdf
  http://xml.coverpages.org/namesAndAddresses.html
  http://xml.coverpages.org/ADIS-Address-2001-1.pdf
- Unicode Technical Standard #35 Unicode Locale Data Markup Language (LDML)
  http://unicode.org/reports/tr35/

Regulatory Standards

- OECD Standard for automatic exchange of financial account information in tax matters
  http://www.oecd.org/tax/exchange-of-tax-information/standard-for-automatic-exchange-of-financial-account-information-in-tax-matters-second-edition-9789264267992-en.htm
- OECD Common Reporting Standard XML Schema
  http://www.oecd.org/tax/automatic-exchange/common-reporting-standard/schema-and-user-guide/#d.en.345315
- ICAO Travel Document Standard – Doc 9303
  https://www.icao.int/publications/pages/publication.aspx?docnum=9303

- UK Deed Poll name change requirements
  https://www.gov.uk/government/publications/change-of-name-guidance/use-and-change-of-names#grounds-for-refusing-to-change-a-name-on-a-home-office-issued-document

Industry Standards

- Google Person / Name Object
  https://developers.google.com/people/api/rest/v1/people#name
- Apple CNMutableContact person object
  https://developer.apple.com/documentation/contacts/cnmutablecontact
- Apple person name components formatter
  https://developer.apple.com/documentation/foundation/personnamecomponentsformatter
- Facebook User Object
  https://developers.facebook.com/docs/graph-api/reference/user/#fields
- Microsoft Personal Contact Graph REST API
  https://docs.microsoft.com/en-us/graph/api/resources/contact?view=graph-rest-1.0#properties
- Trulio Identity Schema
  https://developer.trulioo.com/reference#identity-verification-verify
- Workday Person_Name_Detail_Data
  https://community.workday.com/sites/default/files/file-hosting/productionapi/Human_Resources/v35.0/Change_Legal_Name.html#Person_Name_Detail_DataType

Other References

- Wikipedia
    - Personal Name
      https://en.wikipedia.org/wiki/Personal_name
      https://en.wikipedia.org/wiki/Given_name
      https://en.wikipedia.org/wiki/Middle_name
      https://en.wikipedia.org/wiki/Surname
    - Wikipedia, Naming Conventions
      https://en.wikipedia.org/wiki/Wikipedia:Naming_conventions_(people)
    - Wikipedia Manual of Style, Biographical names
      https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Biography#Names

    - Wikidata personal name object
      https://www.wikidata.org/wiki/Q1071027
- Graham Rhind, Global Sourcebook for International Data Management
  https://www.grcdi.nl/gsb/global%20sourcebook.html
  https://www.grcdi.nl/gsb/world%20personal%20name%20formats.html
  https://www.grcdi.nl/gsb/muslim%20personal%20names%20%2D%20a%20guide.html
- A Guide to Names and Naming Practices
  https://www.fbiic.gov/public/2008/nov/Naming_practice_guide_UK_2006.pdf
- Law Enforcement Guide to International Names
  https://info.publicintelligence.net/ROCICInternationalNames.pdf
- Pan-Data: Names
  https://github.com/pan-i18n/pan-data/blob/master/specs/Names.md