Processing device variations and substitutions

String Catalogs

With the introduction of the String Catalogs in Xcode 15, developers can now specify device variation and substitution rules using a user-friendly UI. Those rules, contrary to the simple plural rules, are not natively supported by the Transifex web interface.

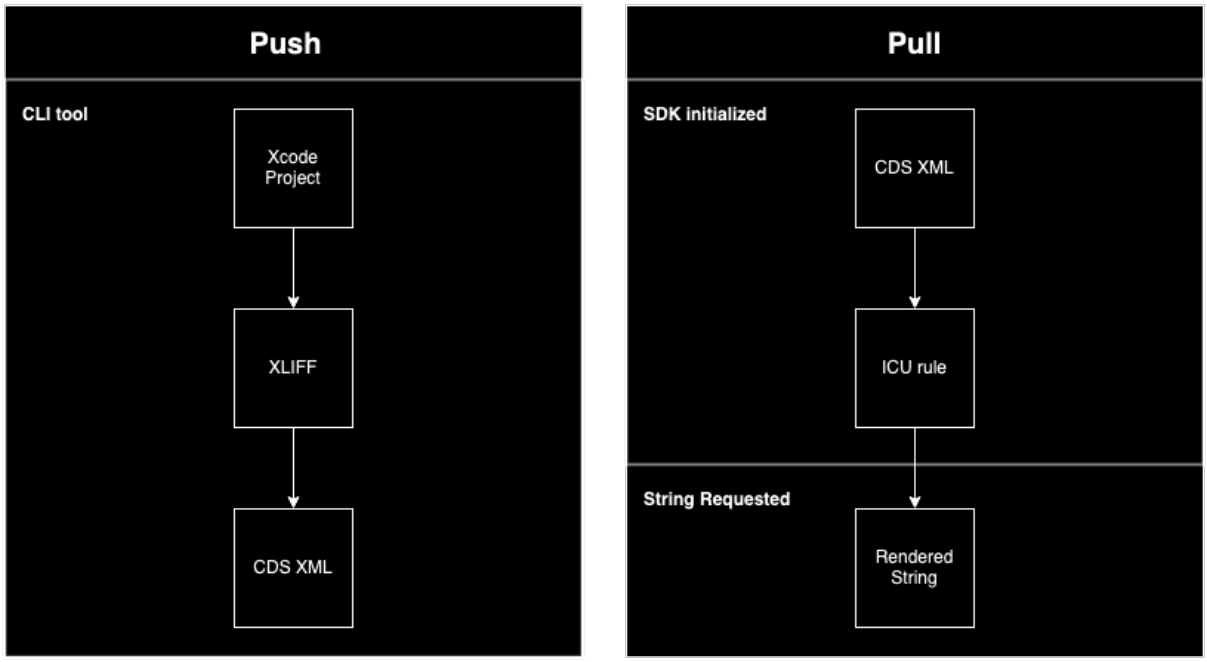
The upcoming versions of the Transifex Native iOS SDK as well as the accompanying CLI tool, have implemented a way to allow those rules to be extracted, processed and rendered in the application.

In order for that to happen, the CLI tool now detects the new rules and produces a XML structure, that is presented in the web interface, allowing translators to localize the strings with ease.

The reason the CLI tool needs to convert those rules into this intermediate XML structure, is so that the parsed rules can be represented in the web interface while maintaining the extra metadata (plural rules) needed for their reconstruction later on.

After the localization has been completed and the strings are pulled from the server by the application, the SDK parses those XML structures and synthesizes an ICU rule during the SDK initialization (when the cache is populated). This step is needed so that the XML parsing does not occur when a string is requested, potentially impacting performance.

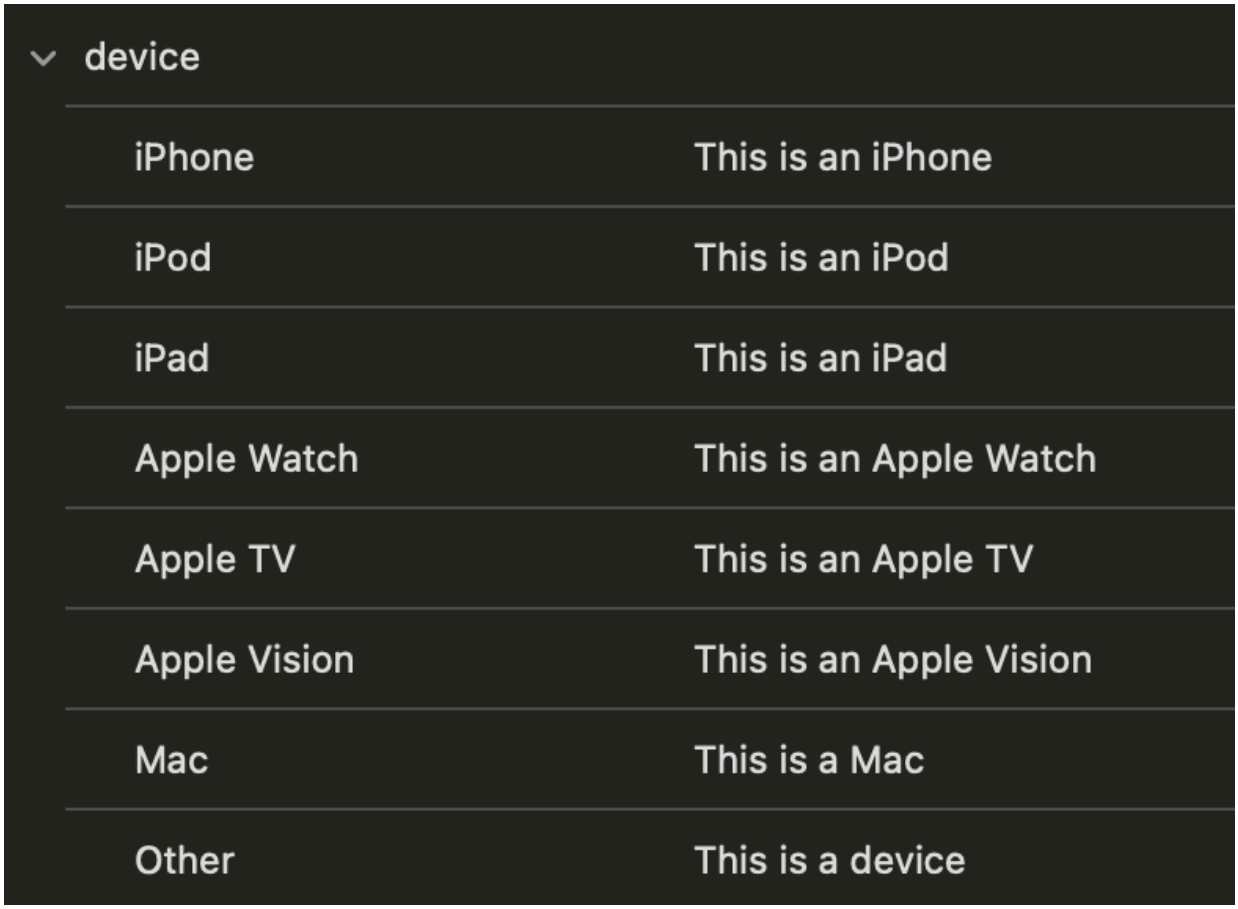
This synthesized ICU rule is then ready to be used when requested by the UI logic of the application, alongside the argument(s) that the application will provide for the pluralizaation.



Below are some examples of this process.

Example 1: Simple device rule

Developer decides to have a label on their app, showing the current device type. For that, it creates a new string in the String Catalog, with the key 'device' and varies the values by devices like so:



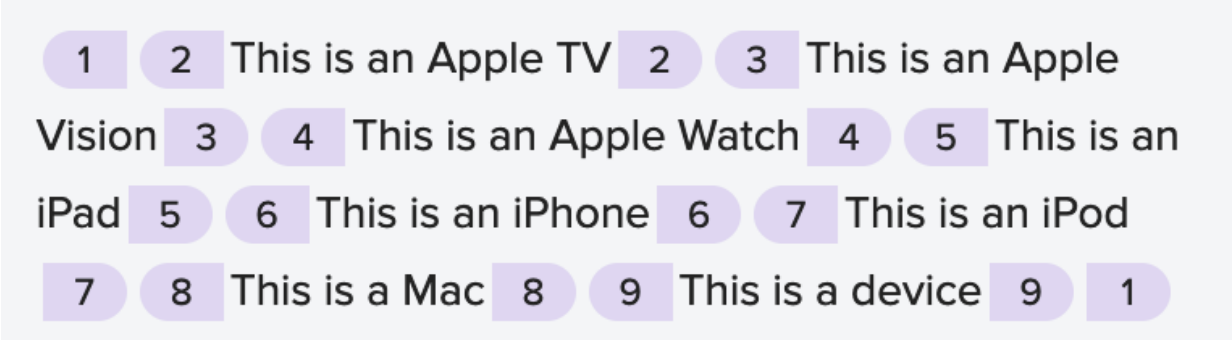
When this string is exported for localization, the intermediate `.xliff` representation will be something like the following:

```
<trans-unit id="device|==|device.appletv" xml:space="preserve">
    <source>This is an Apple TV</source>
    <target state="translated">This is an Apple TV</target>
    <note/>
</trans-unit>
<trans-unit id="device|==|device.applevision" xml:space="preserve">
    <source>This is an Apple Vision</source>
    <target state="translated">This is an Apple Vision</target>
    <note/>
</trans-unit>
<trans-unit id="device|==|device.applewatch" xml:space="preserve">
    <source>This is an Apple Watch</source>
    <target state="translated">This is an Apple Watch</target>
    <note/>
</trans-unit>
<trans-unit id="device|==|device.ipad" xml:space="preserve">
    <source>This is an iPad</source>
    <target state="translated">This is an iPad</target>
    <note/>
</trans-unit>
<trans-unit id="device|==|device.iphone" xml:space="preserve">
    <source>This is an iPhone</source>
    <target state="translated">This is an iPhone</target>
    <note/>
</trans-unit>
<trans-unit id="device|==|device.ipod" xml:space="preserve">
    <source>This is an iPod</source>
    <target state="translated">This is an iPod</target>
    <note/>
</trans-unit>
<trans-unit id="device|==|device.mac" xml:space="preserve">
    <source>This is a Mac</source>
    <target state="translated">This is a Mac</target>
    <note/>
</trans-unit>
<trans-unit id="device|==|device.other" xml:space="preserve">
    <source>This is a device</source>
    <target state="translated">This is a device</target>
    <note/>
</trans-unit>
```

When using the Transifex CLI tool, the above representation is not visible to the developer. The CLI tool transforms the above representation to the following format and pushes it to CDS:

```
<cds-root>
    <cds-unit id="device.appletv">This is an Apple TV</cds-unit>
    <cds-unit id="device.applevision">This is an Apple Vision</cds-unit>
    <cds-unit id="device.applewatch">This is an Apple Watch</cds-unit>
    <cds-unit id="device.ipad">This is an iPod</cds-unit>
    <cds-unit id="device.iphone">This is an iPhone</cds-unit>
    <cds-unit id="device.ipod">This is an iPad</cds-unit>
    <cds-unit id="device.mac">This is a Mac</cds-unit>
    <cds-unit id="device.other">This is a device</cds-unit>
</cds-root>
```

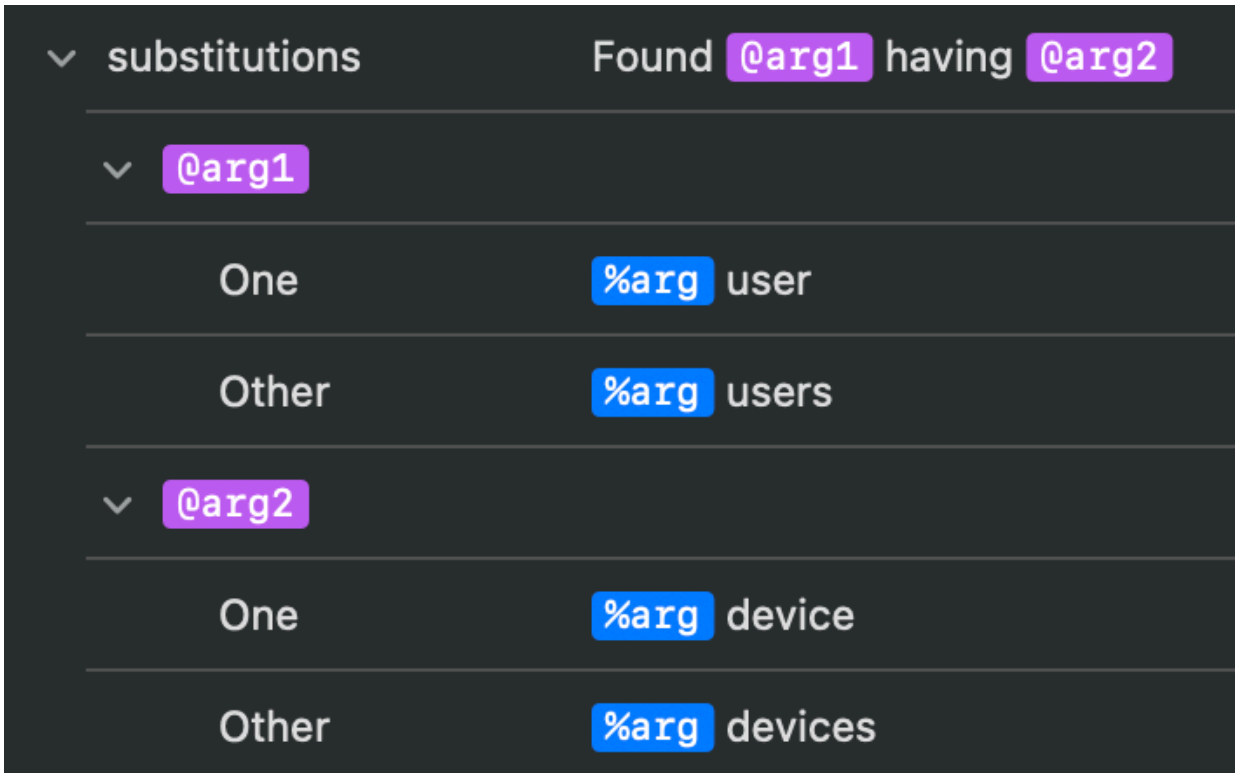
This structure is then displayed within the Transifex web interface, in a way that prevents translators from changing the XML tag names and attributes:



Upon pulling the translated strings, the SDK parses this XML structure and picks the proper tag based on the current device type.

Example 2: Substitutions

Similar to the above example, developer can create a string in the String Catalog that features two (or more) different tokens, that based on the number that is passed during rendering for each token, it can display a different pluralization rule:



In the intermediate `.xliff` representation, the above rule becomes:

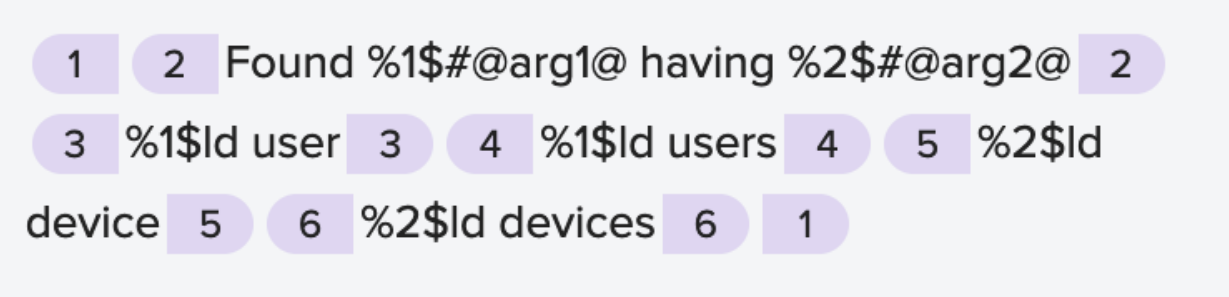
```
<trans-unit id="substitutions" xml:space="preserve">
    <source>Found %1$#@arg1@ having %2$#@arg2@</source>
    <target state="translated">Found %1$#@arg1@ having %2$#@arg2@</targe
    <note/>
</trans-unit>
<trans-unit id="substitutions|==|substitutions.arg1.plural.one" xml:space
    <source>%1$ld user</source>
    <target state="translated">%1$ld user</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions|==|substitutions.arg1.plural.other" xml:sp
    <source>%1$ld users</source>
    <target state="translated">%1$ld users</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions|==|substitutions.arg2.plural.one" xml:space
    <source>%2$ld device</source>
    <target state="translated">%2$ld device</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions|==|substitutions.arg2.plural.other" xml:sp
    <source>%2$ld devices</source>
    <target state="translated">%2$ld devices</target>
    <note/>
</trans-unit>
```

Using the CLI tool, the above rule is transformed in the following XML structure and is passed to CDS:

```
<cds-root>
    <cds-unit id="substitutions">Found %1$#@arg1@ having %2$#@arg2@</cds-
    <cds-unit id="substitutions.arg1.plural.one">%1$ld user</cds-unit>
    <cds-unit id="substitutions.arg1.plural.other">%1$ld users</cds-unit>
    <cds-unit id="substitutions.arg2.plural.one">%2$ld device</cds-unit>
    <cds-unit id="substitutions.arg2.plural.other">%2$ld devices</cds-un
</cds-root>
```

Notice that the key in the main phrase (`Found %1$#@arg1@ having %2$#@arg2@`) is picked by the developer, so the `id` here most likely will not be `substitutions` . The SDK always sets it to "substitutions" though so that the key information is not encoded inside the source string contents.

The web interface then displays the structure like so:



When the localized strings of that rule are pulled and parsed by the SDK, the SDK transforms the XML structure in one rule that contains multiple ICU rules:

```
Found %1$#@{arg1, plural, one {%ld user} other {%ld users}}@ having
%2$#@{arg2, plural, one {%ld device} other {%ld devices}}@
```

The above rule is then used whenever this string is about to be rendered in the UI. The positional specifiers (`1$` , `2$` and so on) are left intact so that the logic can pick the proper argument for the proper rule.

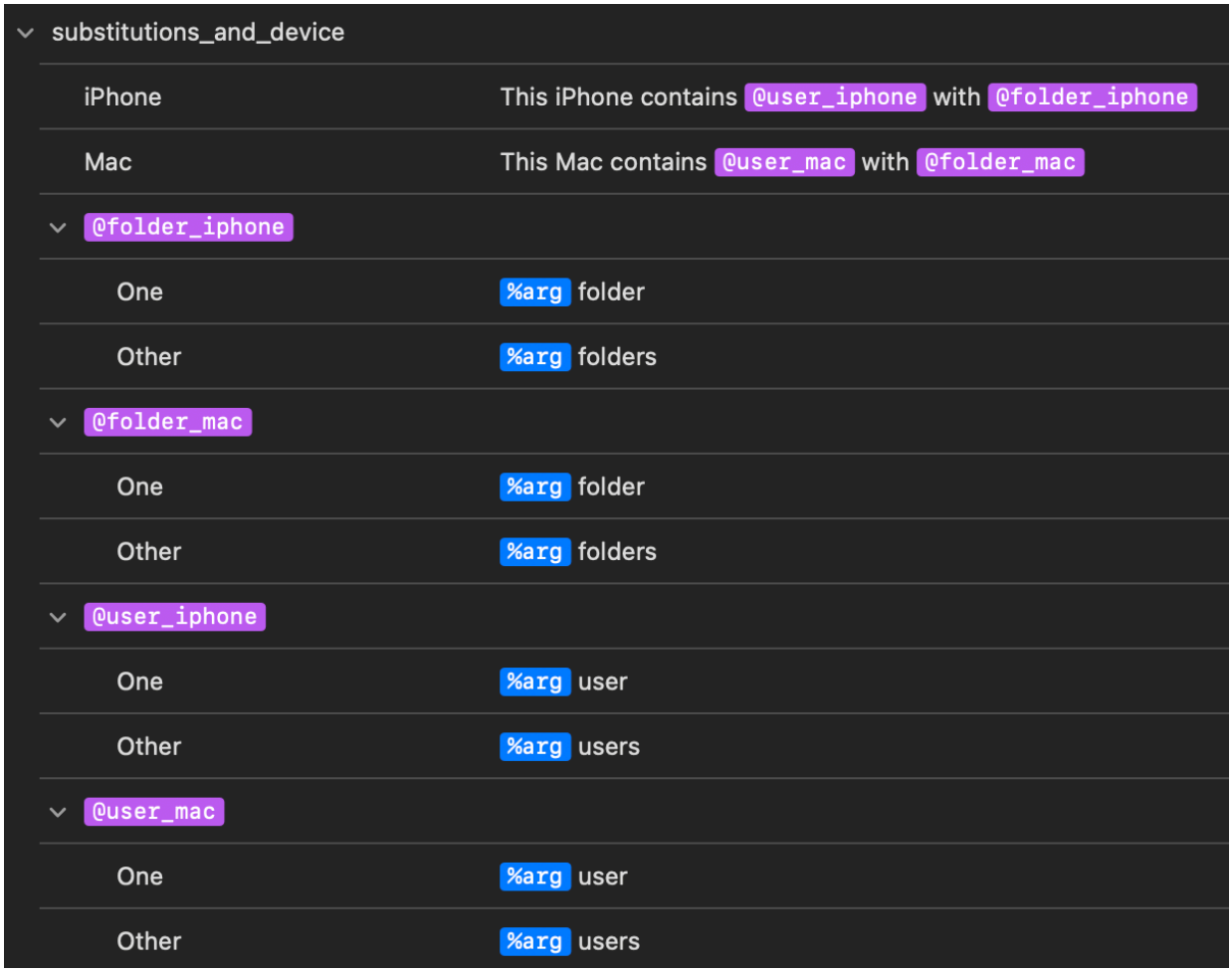
When this string is about to be presented to the user, with the arguments `1` and `10` passed by the applicaton logic, then the SDK reads those arguments, picks up the proper rule, and constructs the final string:

```
Found 1 user having 10 devices
```

Example 3: More complex rules

More complex rules can be constructed by the developer inside the String Catalog: A string can feature substitutions on top of device variations or device variations on plural rules. All those cases are handled correctly by the SDK.

Here is an example of a substitution rule with device variation:



As with the above, the intermediate `.xliff` representation is the following:

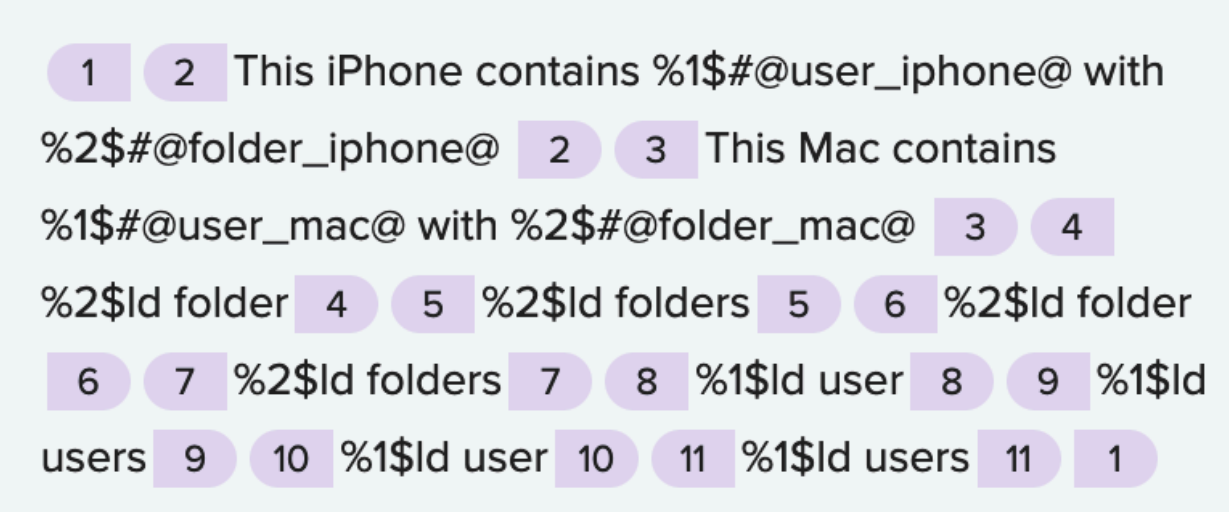
```
<trans-unit id="substitutions_and_device|==|device.iphone" xml:space="pr
    <source>This iPhone contains %1$#@user_iphone@ with %2$#@folder_iphor
    <target state="translated">This iPhone contains %1$#@user_iphone@ wi
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|device.mac" xml:space="prese
    <source>This Mac contains %1$#@user_mac@ with %2$#@folder_mac@ </sou
    <target state="translated">This Mac contains %1$#@user_mac@ with %2$
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|substitutions.folder_iphone.
    <source>%2$ld folder</source>
    <target state="translated">%2$ld folder</target>
    <note/>
</trans-unit>
```

```
<trans-unit id="substitutions_and_device|==|substitutions.folder_iphone.p
    <source>%2$ld folders</source>
    <target state="translated">%2$ld folders</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|substitutions.folder_mac.plu
    <source>%2$ld folder</source>
    <target state="translated">%2$ld folder</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|substitutions.folder_mac.plu
    <source>%2$ld folders</source>
    <target state="translated">%2$ld folders</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|substitutions.user_iphone.plu
    <source>%1$ld user</source>
    <target state="translated">%1$ld user</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|substitutions.user_iphone.plu
    <source>%1$ld users</source>
    <target state="translated">%1$ld users</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|substitutions.user_mac.plural
    <source>%1$ld user</source>
    <target state="translated">%1$ld user</target>
    <note/>
</trans-unit>
<trans-unit id="substitutions_and_device|==|substitutions.user_mac.plural
    <source>%1$ld users</source>
    <target state="translated">%1$ld users</target>
    <note/>
</trans-unit>
```

The above rules are transformed to the following XML structure by the CLI tool and pushed to CDS:

```
<cds-root>
    <cds-unit id="device.iphone">This iPhone contains %1$#@user_iphone@ v
    <cds-unit id="device.mac">This Mac contains %1$#@user_mac@ with %2$#(
    <cds-unit id="substitutions.folder_iphone.plural.one">%2$ld folder</(
    <cds-unit id="substitutions.folder_iphone.plural.other">%2$ld folders
    <cds-unit id="substitutions.folder_mac.plural.one">%2$ld folder</cds-
    <cds-unit id="substitutions.folder_mac.plural.other">%2$ld folders</(
    <cds-unit id="substitutions.user_iphone.plural.one">%1$ld user</cds-u
    <cds-unit id="substitutions.user_iphone.plural.other">%1$ld users</c(
    <cds-unit id="substitutions.user_mac.plural.one">%1$ld user</cds-unit
    <cds-unit id="substitutions.user_mac.plural.other">%1$ld users</cds-u
</cds-root>
```

The web interface displays the above structure like so:



When the localized strings of that rule are pulled and parsed by the SDK, the SDK picks up the correct rule based on the current device (in this example the current device is a Mac) and transforms the XML structure in one rule that contains multiple ICU rules:

```
This Mac contains %1$#@{user_mac, plural, one {%ld user} other {%ld
users}}@ with %2$#@{folder_mac, plural, one {%ld folder} other {%ld
folders}}@
```

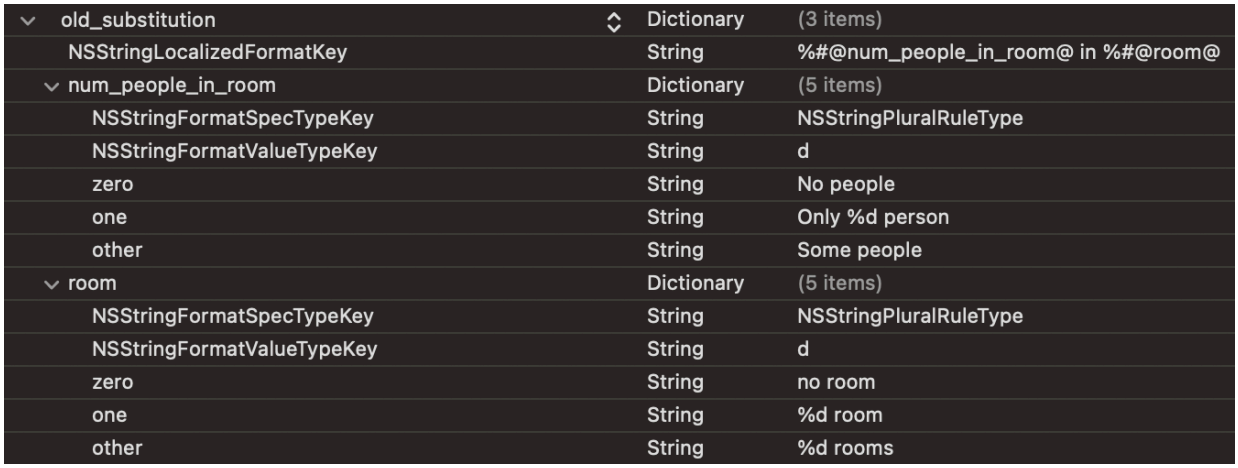
When the string is about to be rendered in the UI, if the arguments are 1 and 5 the final rendered string becomes:

```
This Mac contains 1 user with 5 folders
```

Strings Dictionary Files

One advantage from the support of the above rules, is that support for substitution rules has

been also introduced for the old `.stringsdict` file format, which is the most common format used by developers right now.



old_substitution	Dictionary	(3 items)
NSStringLocalizedFormatKey	String	%#@num_people_in_room@ in %#@room@
num_people_in_room	Dictionary	(5 items)
NSStringFormatSpecTypeKey	String	NSStringPluralRuleType
NSStringFormatValueTypeKey	String	d
zero	String	No people
one	String	Only %d person
other	String	Some people
room	Dictionary	(5 items)
NSStringFormatSpecTypeKey	String	NSStringPluralRuleType
NSStringFormatValueTypeKey	String	d
zero	String	no room
one	String	%d room
other	String	%d rooms

The intermediate `.xliff` representation is the following:

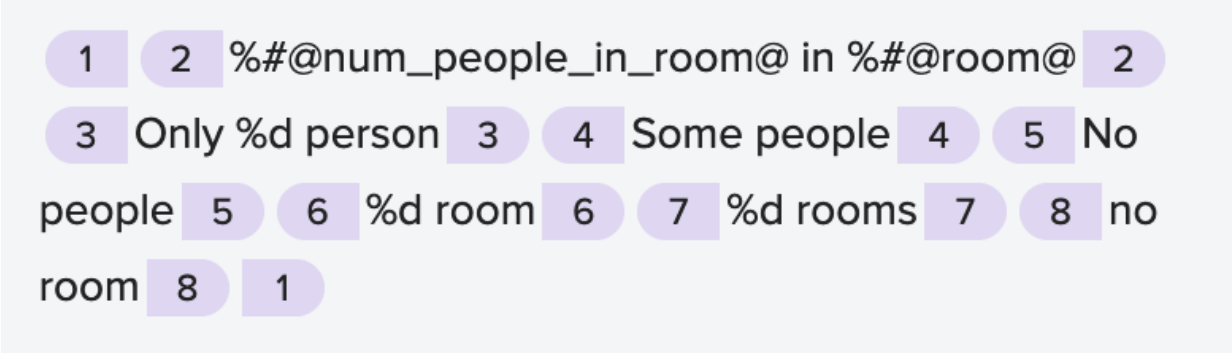
```
<trans-unit id="/old_substitution:dict/NSStringLocalizedFormatKey:dict/:s
    <source>%#@num_people_in_room@ in %#@room@</source>
    <target>%#@num_people_in_room@ in %#@room@</target>
    <note/>
</trans-unit>
<trans-unit id="/old_substitution:dict/num_people_in_room:dict/one:dict/
    <source>Only %d person</source>
    <target>Only %d person</target>
    <note/>
</trans-unit>
<trans-unit id="/old_substitution:dict/num_people_in_room:dict/other:dic
    <source>Some people</source>
    <target>Some people</target>
    <note/>
</trans-unit>
<trans-unit id="/old_substitution:dict/num_people_in_room:dict/zero:dict/
    <source>No people</source>
    <target>No people</target>
    <note/>
</trans-unit>
<trans-unit id="/old_substitution:dict/room:dict/one:dict/:string" xml:sp
    <source>%d room</source>
    <target>%d room</target>
    <note/>
</trans-unit>
<trans-unit id="/old_substitution:dict/room:dict/other:dict/:string" xml
    <source>%d rooms</source>
    <target>%d rooms</target>
    <note/>
</trans-unit>
<trans-unit id="/old_substitution:dict/room:dict/zero:dict/:string" xml:s
    <source>no room</source>
    <target>no room</target>
    <note/>
</trans-unit>
```

The above rules are transformed to the following XML structure by the CLI tool and pushed to CDS:

```
<cds-root>
    <cds-unit id="substitutions">%#@num_people_in_room@ in %#@room@</cds-
    <cds-unit id="substitutions.num_people_in_room.plural.one">Only %d pe
    <cds-unit id="substitutions.num_people_in_room.plural.other">Some peo
    <cds-unit id="substitutions.num_people_in_room.plural.zero">No people
    <cds-unit id="substitutions.room.plural.one">%d room</cds-unit>
    <cds-unit id="substitutions.room.plural.other">%d rooms</cds-unit>
    <cds-unit id="substitutions.room.plural.zero">no room</cds-unit>
</cds-root>
```

Notice how the `old_substitutions` key was renamed to `substitutions`, so that it behaves in the same manner as the String Catalogs approach.

The web interface displays the structure like so:



When the localized strings of that rule are pulled and parsed by the SDK, the SDK transforms the XML structure in one rule that contains multiple ICU rules:

```
%1$#@{num_people_in_room, plural, one {Only %d person} other {Some
people} zero {No people}}@ in %2$#@{room, plural, one {%d room} other
{%d rooms} zero {no room}}@
```

Notice how this intermediate ICU rule includes positional specifiers that are not added by the String Dictionary format. Those specifiers are really important so that the proper argument is picked up.

If the arguments for that string are 3 and 4 then the final string becomes:

```
Some people in 4 rooms
```