# MessageFormat Syntax

*This document is published to the web in the public* [*Angular Design Docs*](https://drive.google.com/#folders/0BxgtL8yFJbacUnUxc3l5aTZrbVk)*/*[*i18n*](https://drive.google.com/folderview?id=0BxyFnbmFLIVxfkpVQzMtcjlTb00zM0xaS0dISXZ4RllnSmp4NGJ6WmNsS21SOFM5ckltZjQ) *folder*

*Chirayu Krishnappa <chirayu@google.com>  
Igor Minar <iminar@google.com>  
Status: Draft  
  
Last updated: Feb 23, 2015*

Links

* <http://plnkr.co/edit/QBVRQ70dvKZDWmHW9RyR?p=preview>
* <https://github.com/angular/angular.js/pull/11152>

## Background

This document discusses the syntax for pluralization and gender based message formatting based on the ICU MessageFormat syntax. As a prerequisite, please read the ICU user guide page on [Formatting Messages](http://userguide.icu-project.org/formatparse/messages#TOC-MessageFormat) and review the EBNF like grammar at [Patterns and Their Interpretation](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/MessageFormat.html#patterns).

## Why messageformat?

* Standard syntax. There exist translators who already understand this syntax and know how to translate these messages.
* Compact syntax allows developers to use the same easy syntax in Angular templates.
* Can be used inside attributes.
* Supports nesting of plurals and gender selections.

## Some general concerns

* Uses # in a message to format the count for a plural. This makes nesting plurals hard. However, since one can perform simple interpolation, you could use that syntax instead of # to nest plural messages as long as you don't have a non-zero offset. (Note, however, that in some languages, the number of choices are reasonably big that you get a huge number of messages when you nest plurals and it's not generally recommended.) As a side effect of the # character, you must now escape the # character with a single quote.
* Awkward escape character. The escape character is the character ' (ASCII 27). Escape it by doubling it.

## Angular specific concerns

### argNameOrNumber is restricted to Pattern\_Syntax

The argument syntax—{NAME}, {NAME, plural, …}, {NAME, select, …}, etc—the NAME is restricted to be a word character (specifically Unicode[Pattern\_Syntax](http://unicode.org/cldr/utility/list-unicodeset.jsp?a=%5B%3APattern_Syntax%3A%5D&abb=on&g=)). This means that NAME cannot be an Angular expression. You can't write user.gender, recipients.length, etc. there. We need to support some way of adding some additional flexibility there.

### "overloading" of the { operator

First, here's an overview of a simple conceptual model that treats { as switching between "plain text" and "interpreted" modes of the syntax (i.e. the syntax is modal and has exactly two modes and you can switch between them using {—and those can be nested arbitrarily. The grammar rules just work out.)

This is both clever and simple if one only lives in the MessageFormat world.

* syntax: Hello, world
  + No interpolate or formatting. This is plain text. If you parse it as MessageFormat, then the entire string is part of messagetext (from the grammar.)
  + Think of this as our syntax being modal and we were in the "plain text" mode.
* syntax: Hello, {NAME}
  + Going by our modal example, the initial "Hello, " was read in "plain text" mode. Upon encountering the {, we switch into "interpreted mode". Here, until the matching }, we are going to interpret one of simple interpolation, plural or select. (There are more but let's just stick to this for simplicity.)
  + Think of each nested { as switching from plain text mode to interpreted mode or vice-versa.
* syntax: Hello, {GENDER, select, female {Hi gal} male {Hi guy} other {Hi you}
  + Here, going with the modal concept, the {'s after male, female and other are switching back from interpreted mode to plain text mode making "Hi gal", etc. plain text messages (messagetext).
  + If you encounter a { inside such messages, you are switching again from plain text mode to interpreted mode until the corresponding } is encountered.

Now, if you wanted to interpolate inside one of the messages right at the beginning you would have typed a "double curly" ({{).

e.g. Hello, {GENDER, select, female {{NAME}! How are you?} … }

#### How does this relate to Angular?

Angular already uses the double curly syntax to denote interpolation. Given the MessageFormat limitation that argNameOrNumber (i.e. the first word after the opening { in the argument such as GENDER, NAME, etc.) can only be composed of word characters, we could offer full Angular expression flexibility by offering to use double curlies—but then we have conflicting ways to interpret it as a double curly in MessageFormat is the equivalent of switching modes two times—something totally different. Further, inside an Angular expression, a single curly is represents the start of an object literal while MessageFormat would cause a mode switch. Also, we will easily be dealing with triple curlies that are uglier.

Further, the MessageFormat single curly syntax means:

* we have to choose between Angular expression syntax and the MessageFormat syntax or their hybrid when writing a message.
* it *isn't backwards compatible*—we may start recognizing MessageFormat like syntax in regular template HTML that was not meant to be recognized as such.
* Current tooling and workflows understand that the double curlies/mustache is an expression context and not pure text. This information might be used in security audits, grep's etc. Not using the same syntax to support MessageFormat breaks this.

### Embedding HTML inside messageText

MessageFormat itself does not care. However, if we're writing MessageFormat inside our Angular templates and the text contains HTML markup, then these become nodes in the DOM and the compiler, at runtime, does not really see the string that the user wrote but a fragment of the MessageFormat syntax followed by actual DOM nodes interspersed with more fragments and will need to reconstruct the original out of it. We can do that, but this section is here just to call it out to the reader's attention.

## Syntax Proposals

First, let's use an example in pure MessageFormat that we can then try and express in our different syntax proposals.

Assume that on our scope, we have these two objects:

* **user**: the logged in user
* **recipients**: array of user objects—the logged in user has given a gift to this list of users.

Since MessageFormat does not support expressions, we first perform the following assignments:

scope.USER = scope.user.name;  
scope.NUM\_RECIPIENTS = scope.recipients.length;  
scope.RECIPIENT\_GENDER = scope.recipients.length == 1 ? scope.recipients[0].gender : **null**;

Now for the MessageFormat syntax:

{NUM\_RECIPIENTS, plural,  
 =0 {You, {USER}, have not given any gifts.}  
 =1 {RECIPIENT\_GENDER, select,  
 female {You, {USER}, gave her a gift.}  
 male {You, {USER}, gave him a gift.}  
 other {You, {USER}, gave them a gift.}  
 }  
 other {You gave # people gifts.}  
 }

That's a hypothetical example but used just to illustrate nesting messages (only one of the messages of plural is nesting another syntax—for gender.)

## Observation 1

The grammar for Angular expressions disallows the "comma" operator (like that which might be used for tuples or destructuring—not as used to separate items of an argument list, array/object, etc.)

e.g. This is an invalid Angular expression: a, b

This means that:

* We can naturally extend existing Angular expressions to have a trailing comma followed by one of select, plural, etc. followed by another comma the the arguments for that. There is no ambiguity here. This obviously doesn't work for nesting and also doesn't allow messages to have interpolations (we'll cover these later in this document.)
* We can alternatively extend MessageFormat such that the argNameOrNumber that was restricted to a word can be any Angular expression without issues. This supports nesting without a hitch—however, messages can't use Angular expressions for interpolation.

## Syntax proposal 1: MessageFormat inside Angular expressions

An example will illustrate this quickly.

{{recipients.length, plural,  
 =0 {You, {{user.name}}, have not given any gifts.}  
 =1 {recipients[0].gender, select,  
 female {You, {{user.props[0]}}, gave her a gift.}  
 male {You, {{user.name}}, gave him a gift.}  
 other {You, {{user.name}}, gave them a gift.}  
 }  
 other {You gave {{recipients.length}} people gifts.}  
 }}

The syntax basically:

* just wraps the outermost single curlies of MessageFormat in curlies to make them double curlies, and
* allows argNameOrNumber to be an arbitrary Angular expression.

However, once inside the outermost double curlies, when not parsing an Angular expression, the { switches MessageFormat modes as in the original MessageFormat syntax. (ref. user.name, recipients[0].gender in the example.)

## Syntax proposal 2: Embed MessageFormat inside an Angular expression

Again, an example will illustrate this nicely.

{{ {  
 NUM\_RECIPIENTS: recipients.length,  
 USER: user.name,  
 RECIPIENT\_GENDER: (recipients.length == 1 ? recipients[0].gender : null)  
 },  
 {NUM\_RECIPIENTS, plural,  
 =0 {You, {USER}, have not given any gifts.}  
 =1 {RECIPIENT\_GENDER, select,  
 female {You, {USER}, gave her a gift.}  
 male {You, {USER}, gave him a gift.}  
 other {You, {USER}, gave them a gift.}  
 }  
 other {You gave # people gifts.}  
 }  
}}

In this syntax, we don't mess with MessageFormat at all.

Like in the previous syntax, we're using a comma after the expression.

The expression begins with an Angular expression that is an object literal, followed by a comma, then the full original MessageFormat syntax.

The keys of this object literal are the only names that can be used in the following MessageFormat for MessageFormat interpolation.

## Performance impacts, limitations, mitigations, etc.

In theory, both syntax proposals would just work.

We want the parser to be both performant and have little to no overhead if we aren't using MessageFormat.

One way to help this is to prefix the Angular expression with something illegal to denote that we're using one of these syntaxes. For instance, we can use = or ? since no expression can begin with them.

e.g. {{**?**recipients.length, plural, … for the first syntax or {{**?**{NUM\_RECIPIENTS: recipients.length, … for the second syntax.

## The Angular 1.4 parser/lexer integration

The Angular parser is fairly easy to extend to support either syntax with minimal changes.

The lexer, however, as I just discovered, is a whole different beast. The way the parser works, is that is first lexes the entire text given to it, and then parses the tokens. This is the traditional way. (PEG like parsers or a those that combine parsing and lexing in lockstep can work around this issue.)

I have some ideas on how to work around this so this may not be a big deal—but just calling it our to your attention, dear reader.

## Writing MessageFormat in document body (i.e. outside attributes.)

Earlier in this document, I wrote the following.

### Embedding HTML inside messageText

*MessageFormat does not care if you embed HTML in the* messageText*. However, if we're writing MessageFormat inside our Angular templates and the text contains HTML markup, then these become nodes in the DOM and the compiler, at runtime, does not really see the string that the user wrote but a fragment of the MessageFormat syntax followed by actual DOM nodes interspersed with more fragments and will need to reconstruct the original out of it.*

We can offer an alternate syntax for this which looks and behaves more like HTML.

e.g.

**<ng-msg** plural="recipients.length"**>**  
 **<ng-when** value="=0"**>**  
 You, {{user.name}}, have not given any gifts.  
 **</ng-when>**  
 **<ng-when** value="=1"**>**  
 **<ng-msg** select="recipients[0].gender"**>**  
 **<ng-when** value="female"**>**  
 You, <b>{{user.name}}</b>, gave her a gift.  
 **</ng-when>**  
 **<ng-when** value="male"**>**  
 You, {{user.name}}, gave him a gift.  
 **</ng-when>**  
 **<ng-when** value="other"**>**  
 You, {{user.name}}, gave them a gift.  
 **</ng-when>**  
 **</ng-msg>**  
 **</ng-when>**  
 **<ng-when** value="other"**>**  
 You, {{user.name}}, gave {{recipients.length}} people gifts.  
 **</ng-when>**  
**</ng-msg>**

This is much more verbose, but also much clearer.

Please note that this syntax is **recognized during precompilation** and will require a strict syntax where the only child nodes of msg are whennodes, etc. (*Actual tag names TBD as this syntax is not required for Angular 1.4.*)