# Ship i18n v1.0

*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 09, 2015*

## Goal

Ship i18n v1.0 integrated with Angular 1.4.

## Timeline

Merge with AngularJS master by Feb 20th. (AngularJS v1.4 is scheduled to be released around the end of February.)

This gives us roughly about a month and a half.

[Ship i18n v1.0](#h.ixg45w3363q)

[Goal](#h.yomaneot3qhc)

[Timeline](#h.oydqzp8h3yh7)

[What's needed for v1.0?](#h.ay91f91c1oaw)

[Constraints](#h.wv3a2h7jzvtw)

[Backwards compatibility](#h.svx1z8ysxpyr)

[Stable new syntax](#h.itngdyetfbp2)

[Tools and Libraries](#h.cf33m634al6t)

[Template Discovery](#h.3q4in5h88svp)

[Offline Message Extraction and linting library](#h.8lui7p5n4tkz)

[Linter](#h.1i8t45ypns4y)

[Offline Message Extractor](#h.lsxv7sz1n1g9)

[File formats support](#h.qn6toa6rm08n)

[Offline static application generator](#h.ev8ss64eyc89)

[Angular Core Support](#h.jrcnm71159ia)

[Locale detection](#h.86r8fgf1tp3)

[Online Message Extractor](#h.kd97t1f12dro)

[Pre-compile stage](#h.jlc9v3h78ij1)

[Template Cache integration](#h.7kux0efo9wsm)

[Documentation](#h.4dygf676231d)

[Testing support](#h.rkqwkekpnisr)

[Pseudolocalization and debug modes](#h.n2we1yisyhoa)

[Unittest support](#h.jtpe14ouqhbf)

[Developer tools](#h.kvd06663n31n)

[Post 1.0](#h.tzvz64ohlrk6)

[Tasks for current milestone](#h.490tvyc29lrp)

[Runtime support needed for gender and pluralization](#h.pjpz4diqvb0g)

## What's needed for v1.0?

### Constraints

#### **Backwards compatibility**

Fully backwards compatible with Angular 1.3 for developers not opting in to i18n. It's ok to have some breaking changes for those using i18n if it's hard to avoid.

#### **Stable new syntax**

Lock down the new syntax so that we can continue with i18n implementation in following releases without breaking changes. This means that we should be strict about the syntax that we accept, recognize and throw errors on unimplemented syntax/features, etc.

### Tools and Libraries

#### **Template Discovery**

This library will be used by the linter and message extractor to resolve references to templates. These templates would include those belonging to directives, script[type="ng/template"] blocks as well as the primary entry points of the application.

#### **Offline Message Extraction and linting library**

* This library will be used by the linter and the message extractor.
* Support for implicit syntax. This should be driven by an external specification file.
* Message canonicalization: Functions to canonicalize recognized messages.
* Message ID generation: Functions to construct message IDs from extracted messages.

#### **Linter**

* Strict mode and not so verbose modes.
* Good error messages.

#### **Offline Message Extractor**

* Traverse the source tree and extract i18n messages into our custom JS structures.

#### **File formats support**

* Support for XMB/XTB, XLIFF 1.2 and gettext/po.
* Library to convert between our JS types and these file formats. It is a non-goal to have non-lossy conversion between these formats. The JS types will be the canonical/authoritative syntax.
* Our JS syntax should be designed with compatibility in mind. As folks upgrade to the next version of Angular, they could have older files lying around and we should support reading and processing them.

#### **Offline static application generator**

Generate static versions of the application per locale or locale group.

### Angular Core Support

#### **Locale detection**

Includes support to reload the application upon a locale change.

#### **Online Message Extractor**

Fast and small: This code will run in a pre-compile step during Angular template compilation. It needs to run fast and contribute minimally to the Angular binary.

Timeline caveat: Get implementation started by Feb 1, else cut it out of v1.

#### **Pre-compile stage**

Angular's compiler needs to perform a pre-compile stage while compiling directives. This stage will use the online message extractor to recognize i18n messages, compute the the message ID, lookup the message ID in the translated messages service (JSON file), and rewrite the HTML to use the translated message and call the appropriate functions. (e.g. for pluralization, etc.) Note that this is only applicable to the dynamic mode. In static mode, the pre-compile stage will be skipped. (method TBD—simplest is to mark the template as already translated.)

#### **Template Cache integration**

Integration with the $templateCache service.

### Documentation

* User Guide
* Example applications
* Best Practices document
  + Also call out Bidi changes coming in post 1.0.
* Reference

### Testing support

#### **Pseudolocalization and debug modes**

* In these mode, the user will be served a pseudolocalized application where in English words are fake internationalized (by accenting characters and increasing the word sizes by doubling characters.)
* Modes would be a mix and match of: accent character substitution, word length doubling.
* Debug highlighting: To quickly see the parts of the UI that are and aren't translated.

#### **Unittest support**

* Support for writing unit tests for i18n.

### Developer tools

These are tools that we, as developers working on i18n, need in order to be productive.

* UI (webapp) to display some kind of translator's view of a message.
* Fast full pipeline tests.

## Post 1.0

* Proper Bidi support
  + pseudo localization support for inserting spans.
  + development/testing/protractor support for detecting CSS mismatches by comparing versions with and without extra spans (e.g. viagetComputedStyle())
* IDE and Editor support
  + Webstorm/Jetbrains, Netbeans, Vim, Emacs, Sublime.
* Migration story and tool support
  + for those with legacy apps with no i18n
  + for those using i18n with other existing frameworks (angular-translate, closure-i18n, etc.)
  + partial and wholesale migration tools.

## Tasks for current milestone

Refer to <https://github.com/angular/i18n/issues> for related issues. The message type specification is in [lib/message\_types.ts](https://github.com/angular/i18n/blob/master/lib/message_types.ts)

In a nutshell, this is what we want:

* Specification
  + Specify the JSON serialization format
  + Specification/pseudocode for calculating the message id
  + Specify the runtime support functions needed (ICU/MessageFormat/pluralization and gender)
* Core
  + AtScript version of message\_types.ts
  + Compare two extracted messages for equivalence with extra warning/error details
  + Compare extracted and translated message for equivalence
  + Lint translated file
  + Compute changed/new/deleted messages
  + Function to compute message id based on the specification in pseudocode.
* Config
  + read and validate config from package.json
* AngularJS 1.4
  + $i18n service
    - Default config is to load i18n.json via XHR. fail if not found. great for dev workflow
    - $i18nConfig.urlCombiner = function(left, right) { }
  + Runtime support for ICU MessageFormat (pluralization, gender)
* Serialization
  + Message to/from JSON with tests
  + Message to/from XLIFF with tests
  + Message to/from XTB/XMB with tests
* Infrastructure
  + Build script
    - Transpile to atscript
  + Script to run tests locally
  + Run tests on travis
* Runtime Support
* URL Stuff
* Documentation

### Runtime support needed for gender and pluralization

We transform the MessageFormat syntax into an Angular expression that can work inside of attributes as well.

Here's a sample rewrite for pluralization:

Source: (Angular Template)

**<div** title="{COUNT, plural, =0 {No things} =1 {One thing} other {# things}}"  
 i18n-title="number of things"**>**

Result: (ngI18n directive or localization build step) The developer would never write this, but if you "inspect element", you'll see this.

**<div** title="{{ $$i18n.msgFmtPluralize(  
 COUNT, {  
 0: "No things",  
 1: "One thing",  
 "other": "# things"  
 } }}**>**

The ngPluralize filter can then be implemented in terms of this function.

Likewise for the select filter used for gender.

The runtime support requirements are:

* $$i18n service with
  + msgFmtPluralize()
  + msgFmtSelect()
* The $$i18n service should be available in every scope context. (!)
* ngI18n Directive
  + Runs at very high priority and recognized messageformat syntax and tranforms it into an Angular expression using the $$i18n service from the earlier steps.

Assignee: Pawel or Chirayu: Chirayu will pick this up on Thursday if not already taken.