**How to Localize Your App With jQuery.I18n**

**In this article, learn about the main features of jQuery.i18n library, particularly those that have to do with translations and localization.**

by [Ilya Bodrov](https://dzone.com/users/2885442/bodrovis.html) Dec. 02, 16 · [Web Dev Zone](https://dzone.com/web-development-programming-tutorials-tools-news) · Tutorial

Join the DZone community and get the full member experience.

For the past month, I have written a bunch of articles about [internationalization](https://phraseapp.com/blog/posts/i18n-a-simple-definition/) covering the [usage of I18n in Rails apps](https://phraseapp.com/blog/posts/last-rails-i18n-guide-youll-ever-need/) as well as [various solutions for JavaScript](https://phraseapp.com/blog/posts/rails-i18n-setting-and-managing-locales/). Today's guest is the jQuery.I18n library that was briefly covered in one of the previous posts. This solution supports gender, grammar forms, dynamic change of language, fallback chains, and more.

We are going to thoroughly discuss jQuery.I18n's features as well as see it in action by creating a demo JavaScript application. Here is a quick overview of the themes that we'll cover:

* Basic information about jQuery.I18n.
* Setting up demo application and loading the necessary files.
* The process of creating translation files.
* Loading translation files and some gotchas to take into consideration.
* Switching locale.
* Persisting locale data by using History API.
* Displaying localized messages based on HTML5 data- attributes.
* Displaying localized messages using custom JavaScript code.
* Employing gender and pluralization info in translations.
* Prettifying the code to avoid repeating ourselves.
* Adding "magic" words.
* Documenting your translations.

The source code for this article is available at [GitHub](https://github.com/bodrovis/PhraseApp-jQueryI18n).

**A Quick Note About the Demo App**

To be able to see the final result in action, you'll need to setup a web server. I am going to use [Microsoft IIS](https://www.iis.net/), but there are plenty of other solutions available that can get you up and running in a minute:

* [MAMP](https://www.mamp.info/en/).
* [WAMP](http://wamp.io/).
* And others.

**About jQuery.I18n**

Of course, you know what [Wikipedia](https://www.wikipedia.org/) is and probably has used this website many times, as it has knowledge about anything starting from physics and chemistry to popular (and less popular) films and computer games. Wikipedia is maintained by a company called [Wikimedia Foundation](https://www.wikimedia.org/) that also takes care of projects like [Wikinews](https://www.wikinews.org/), [Wikivoyage](https://www.wikivoyage.org/), and many others. All these projects are worldwide and available in different languages so internationalization is [absolutely crucial](https://phraseapp.com/blog/posts/how-important-is-localization-for-your-business/) for Wikimedia.

Therefore, this company created a popular jQuery-based solution to internationalize JavaScript applications and called it simply jQuery.I18n. Not a very fancy name, but believe me, there is more than meets the eye. This library is really convenient and powerful, so let's not waste our time and discuss its features now. To make our journey more interesting and useful, we are going to apply the learned concepts into practice.

**Translation Files**

jQuery.I18n [stores translations](https://github.com/wikimedia/jquery.i18n#message-file-format) in the simple JSON files. If you are, like me, a fan of Rails, you'll find this process very similar to storing translations in YAML files.

The actual translations are in key-value format, for example

{ "title": "Example Application" }

title is, of course, the key, whereas Example Application is its value. It is advised to name keys in lowercase with words separated by -. jQuery.I18n's docs also suggests to prefix your keys with an application's name:

{ "myapp-title": "Example Application" }

...but that's not mandatory.

Apart from translations, files may host metadata, like information about authors, last updated date etc. Metadata has its own key starting with @:

{

"@metadata": {

"authors": [

"Alice",

"David"

],

"last-updated": "2016-10-25",

"locale": "en"

},

"title": "Example Application"

}

Usually, translation files are being placed in the *i18n* directory. Translations for different languages are often stored separately in the files named after the language (for example *en.json*, *de.json*, *ru.json* and so on). However, for a simple application, you may put [everything in a single file](https://github.com/wikimedia/jquery.i18n#single-message-file-for-all-languages). In this case, translations should be placed under the key named after the language:

"en": {

"title": "Example Application"

},

"ru": {

title: "Тестовое приложение"

}

Moreover, you may provide a path to the file instead of an object with translations:

"en": {

"title": "Example Application"

},

"ru": "ru/ru.json"

To start crafting our demo application, create a new folder with a nested *js/i18n* directory. Place translation files inside for the languages you prefer. I am going to work with English and Russian in this article:

* *js/i18n/en.json*
* *js/i18n/ru.json*

Populate your files with some basic contents:

*js/i18n/en.json*

{

"@metadata": {

"authors": [

"Ilya"

],

"last-updated": "2016-10-25",

"locale": "en"

},

"app-title": "Example Application"

}

*js/i18n/ru.json*

{

"@metadata": {

"authors": [

"Ilya"

],

"last-updated": "2016-10-25",

"locale": "ru"

},

"app-title": "Тестовое приложение"

}

You might need to setup a [proper MIME type](https://stackoverflow.com/questions/477816/what-is-the-correct-json-content-type) for the *.json* extension in your server config.

Also, in order to prepare for the next steps, clone jQuery.I18n locally along with its dependencies:

$ git clone https://github.com/wikimedia/jquery.i18n.git

$ cd jquery.i18n

$ git submodule update --init

Inside your project's directory create *js/lib/jquery.i18n* folder and copy the contents of the *src* folder from the cloned project inside (without the nested *languages* directory). Next inside the *js/lib* create another directory *CLDRPluralRuleParser*. Copy the *libs/CLDRPluralRuleParser/src/CLDRPluralRuleParser.js* file from the cloned project there.

Lastly inside the *js* create a file called *global.js* where we will put our custom code. As a result, your project should have the following structure:

|-- js

|-- global.js

|-- i18n

|-- en.json

|-- ru.json

|-- lib

|-- CLDRPluralRuleParser

|-- CLDRPluralRuleParser.js

|-- jquery.i18n

|-- jquery.i18n.js

|-- jquery.i18n.language.js

|-- ...other files here

Now in the root of the project create a simple HTML file with the necessary libraries hooked up in the proper order:

*demo.html*

<!DOCTYPE>

<html>

<head>

<meta charset="utf-8">

<title>Demo Application</title>

<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>

<script src="js/lib/CLDRPluralRuleParser/CLDRPluralRuleParser.js"></script>

<script src="js/lib/jquery.i18n/jquery.i18n.js"></script>

<script src="js/lib/jquery.i18n/jquery.i18n.messagestore.js"></script>

<script src="js/lib/jquery.i18n/jquery.i18n.fallbacks.js"></script>

<script src="js/lib/jquery.i18n/jquery.i18n.language.js"></script>

<script src="js/lib/jquery.i18n/jquery.i18n.parser.js"></script>

<script src="js/lib/jquery.i18n/jquery.i18n.emitter.js"></script>

<script src="js/lib/jquery.i18n/jquery.i18n.emitter.bidi.js"></script>

<script src="js/global.js"></script>

</head>

<body>

</body>

</html>

I am using jQuery 3 here but [you may choose](https://jquery.com/download/) Version 1 or 2 depending on the browsers you wish to support (Version 3 works only with modern browsers, so no IE7 for you).

So far, so good. We can proceed to the next section and load our translation into the application.

**Loading Translations**

The next important step is, of course, [loading your translations](https://github.com/wikimedia/jquery.i18n#message-loading). In the simplest case, you would just provide the key-value pairs directly in the code:

$.i18n().load( {

'en': {

'app-title': 'Example Application'

}

} );

Or for a specific locale:

$.i18n().load({

'app-title': 'Example Application'

}, 'en');

For complex apps, however, that's not the best choice. Instead, I'd recommend specifying a remote URL like this:

$.i18n().load( {

'en': 'dir/en.json'

} );

Do note that you may load messages in parts, meaning that this code:

$.i18n().load( {

'en': {

'app-title': 'Example Application'

}

} );

// ... some instructions

$.i18n().load( {

'en': {

'another key': 'a value'

}

} );

is totally valid. Just make sure you don't overwrite any existing keys.

Let's load translations into the demo app now:

*js/global.js*

jQuery(function($) {

$.i18n().load( {

'en': './js/i18n/en.json',

'ru': './js/i18n/ru.json'

} );

});

Another important thing is that the load function returns a [jQuery Promise](https://api.jquery.com/promise/), making it easy to perform some action only after the translations are loaded successfully, for example:

$.i18n().load( {

'en': 'i18n/en.json',

'ru': 'i18n/ru.json'

} ).done( function() { console.log('done!') } );

Now that our translations are loaded, we can introduce a mechanism to [switch locales](https://github.com/wikimedia/jquery.i18n#switching-locale).

**Switching Locale**

Which locale is going to be set as a default upon the page load? First of all, the default locale can be provided as an option passed to the $.i18n:

$.i18n( {

locale: 'en'

} );

Also, jQuery.I18n will check the value of the lang attribute set for the html tag. Let's add it now:

*demo.html*

<html lang="en" dir="ltr">

If the default locale was not set in any of these two ways, the library will try to get the language setting passed by the browser. Internally, the default language is set to English. To avoid any unexpected behavior, it is recommended to set the default locale explicitly.

Of course, it is possible to switch the chosen locale later by modifying the locale option:

$.i18n().locale = 'ru';

Let's add links to switch locale now:

*demo.html*

<ul class="switch-locale">

<li><a href="#" data-locale="en">English</a></li>

<li><a href="#" data-locale="ru">Русский</a></li>

</ul>

Handle the click event by preventing the default action and switching locale based on the data-locale attribute:

*js/global.js*

jQuery(function($) {

// ...

$.i18n().load( {

'en': './js/i18n/en.json',

'ru': './js/i18n/ru.json'

} ).done(function() {

$('.switch-locale').on('click', 'a', function(e) {

e.preventDefault();

$.i18n().locale = $(this).data('locale');

});

});

});

Note that I am putting this event handler inside the promise to make sure that the translations are loaded first.

**Persisting Locale**

Another thing that we are going to do is persist the chosen locale by appending it to the URL in a form of a GET param. So, for example, this URL http://localhost/demo.html?locale=ru is going to request the Russian version of the website. It is very important to do so because when a user shares a link he expects that other people will see the same version of the site.

In order to achieve this, however, we'll need two additional libraries. The first one is [History API](https://github.com/browserstate/history.js) that supports [HTML5 History/State APIs](https://developer.mozilla.org/en-US/docs/Web/API/History_API). I am going to take the bundled html4+html5 version for jQuery that can be found [here](https://github.com/browserstate/history.js/blob/master/scripts/bundled/html4%2Bhtml5/jquery.history.js).

The second is [URL](https://github.com/websanova/js-url) library that makes working with URLs in JavaScript a breeze. Grab the minified version [here](https://github.com/websanova/js-url/blob/master/url.min.js).

Hook up these two libraries now:

*demo.html*

<!-- ... -->

<script src="js/lib/history/jquery.history.js"></script>

<script src="js/lib/url/url.min.js"></script>

<script src="js/global.js"></script>

<!-- ... -->

Now code a simple function to change the locale:

*js/global.js*

var set\_locale\_to = function(locale) {

if (locale)

$.i18n().locale = locale;

};

It should be called once the page is loaded and translations are done. The locale itself should be fetched from the ?locale parameter:

*js/global.js*

jQuery(function() {

$.i18n().load( {

// ...

} ).done(function() {

set\_locale\_to(url('?locale'));

// ...

});

});

url('?locale') takes the value of the locale GET parameter. Also, we need to listen for the statechange event (that happens, for example, when the pushState method was invoked) and change the locale accordingly:

*js/global.js*

jQuery(function() {

$.i18n().load( {

// ...

} ).done(function() {

set\_locale\_to(url('?locale'));

History.Adapter.bind(window, 'statechange', function(){

set\_locale\_to(url('?locale'));

});

// ...

});

});

And lastly use pushState once a link to switch the language was clicked. Here is the final version of the script:

*js/global.js*

var set\_locale\_to = function(locale) {

if (locale)

$.i18n().locale = locale;

};

jQuery(function() {

$.i18n().load( {

'en': './js/i18n/en.json',

'ru': './js/i18n/ru.json'

} ).done(function() {

set\_locale\_to(url('?locale'));

History.Adapter.bind(window, 'statechange', function(){

set\_locale\_to(url('?locale'));

});

$('.switch-locale').on('click', 'a', function(e) {

e.preventDefault();

History.pushState(null, null, "?locale=" + $(this).data('locale'));

});

});

});

Even if the locale parameter is missing, we still have the lang="en" set for the html tag.

**Displaying Translations With Data API**

[The easiest way](https://github.com/wikimedia/jquery.i18n#data-api) to display a localized content is by using data- attributes. All you need to do is provide a data-i18n attribute and set the translation's key as its value. For example, let's display our site name:

*demo.html*

<h1 data-i18n="app-title"></h1>

app-title, as you recall, was defined inside the *en.json* file as "app-title": "Example Application" and in *ru.json* as "app-title": "Тестовое приложение".

To actually display the message use the i18n() method without any arguments. This method should be applied to the exact element or to its parent. We can simply say body:

*js/global.js*

var set\_locale\_to = function(locale) {

if (locale) {

$.i18n().locale = locale;

}

$('body').i18n();

};

// ...

In order to provide a fallback text that will be displayed while translations are being loaded, simply place it into the tag:

*demo.html*

<h1 data-i18n="app-title">Example Application</h1>

Now reload your page and observe result!

**Parameters, Gender, and Pluralization in Translations**

In the previous section, we've seen the simplest example of displaying localized messages by using thedata-i18n parameter. Obviously, in many cases, that's not enough. To fetch translation for an arbitrary key, you can use [the following code](https://github.com/wikimedia/jquery.i18n#jqueryi18n-plugin):

$.i18n('some-key');

This way we can, for example, display a welcoming message (with a fallback text):

*demo.html*

<body>

<!-- ... -->

<h2 id="welcome">Welcome</h2>

</body>

Add translations:

*en.json*

"welcome": "Welcome!"

*ru.json*

"welcome": "Добро пожаловать!"

and tweak the code:

*global.js*

var set\_locale\_to = function(locale) {

// ...

$('#welcome').text($.i18n('welcome'));

};

Pretty nice, but it would be better to display a user's name as well. In order to do this, we need to add parameter inside the message. Parameters in jQuery.I18n are being named $1, $2, $3 etc:

*en.json*

"welcome": "Welcome, $1!"

*ru.json*

"welcome": "Добро пожаловать, $1!"

To set parameter's value, simply pass another argument to the $.i18n:

*global.js*

var set\_locale\_to = function(locale) {

// ...

$('#welcome').text($.i18n('welcome', 'John'));

};

If you had two parameters, you would, of course, pass two arguments to the method and the first one will be assigned to the $1 variable, whereas second - to the $2.

Now, what about the [gender info](https://github.com/wikimedia/jquery.i18n#gender)? Suppose, for example, we want to display a bunch of e-mails from different people with a header "You received a new letter from Someone. He/she says:". First of all, add a new markup:

*demo.html*

<!-- ... -->

<div id="letter-1">

<p><em></em></p>

<p>letter's text...</p>

</div>

<div id="letter-2">

<p><em></em></p>

<p>letter's text...</p>

</div>

Now translations:

*en.json*

"letter": "A letter from $1! {{GENDER:$2|He|She}} says:"

*ru.json*

"letter": "Письмо от $1! {{GENDER:$2|Он|Она}} говорит:"

So GENDER: acts as a switch. It receives parameter and chooses one of two options: the first is picked for the male gender, whereas the second - for the female. To be able to use this message, provide male or female as the second argument:

*global.js*

var set\_locale\_to = function(locale) {

// ...

$('#letter-1').find('p > em').text($.i18n('letter', 'Ann', 'female'));

$('#letter-2').find('p > em').text($.i18n('letter', 'Rick', 'male'));

}

Also, let's display how many letters does the user have. For this, we'll need the [PLURAL: switch](https://github.com/wikimedia/jquery.i18n#plurals).

*demo.html*

<p id="letters"></p>

The messages:

*en.json*

"letters": "You have $1 {{PLURAL:$1|letter|letters}}"

*ru.json*

"letters": "У вас $1 {{PLURAL:$1|письмо|писем|письма}}"

The Russian language has more complex pluralization rules so I had to provide an additional option for the PLURAL: switch. Now the code:

*global.js*

var set\_locale\_to = function(locale) {

// ...

$('#letters').text($.i18n('letters', 5));

};

All information about gender and pluralization is stored inside the *jquery.i18n.language.js* file.

**Prettifying the Code**

Everything is working pretty nice, but things are becoming tedious. For each separate translation, I have to write another line of code which is not particularly great. What I want to do is write a cycle that takes each element requiring translation and displays a message inside based on its parameters.

As a first step, modify the markup:

*demo.html*

<!-- ... -->

<h1 class="translate" data-args="app-title">Example Application</h1>

<h2 class="translate" data-args="welcome,John">Welcome</h2>

<p class="translate" data-args="letters,5"></p>

<div>

<p><em class="translate" data-args="letter,Ann,female"></em></p>

<p>letter's text...</p>

</div>

<div>

<p><em class="translate" data-args="letter,Rick,male"></em></p>

<p>letter's text...</p>

</div>

Now for each element with the .translate class we need to take the value of the data-args, split it into an array and pass to the $.i18n method. The only problem is that we don't know how many arguments are going to be passed. That's not a problem however with the [apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply) method:

var set\_locale\_to = function(locale) {

// ...

$('.translate').each(function() {

var args = [], $this = $(this);

if ($this.data('args'))

args = $this.data('args').split(',');

$this.html( $.i18n.apply(null, args) );

};

apply accepts two arguments: the first is the value of this inside the called function and the second is an array representing all the arguments to pass.

Now you may reload the page and observe the result. Of course this solution is not ideal but you can extend it further as needed.

**Magic Words**

PLURAL: and GENDER: are both "magic" words the library supports. You can, however, [introduce new magic words](https://github.com/wikimedia/jquery.i18n#plurals) as needed by extending the $.i18n.parser.emitter:

*global.js*

jQuery(function() {

$.extend($.i18n.parser.emitter, {

sitename: function() {

return "Demo";

}

});

});

So, the sitename is the magic word's name and the corresponding functions returns its value. To fetch this value, use template-like syntax (similar to what [Handlebars](http://handlebarsjs.com/) use): {{SITENAME}} or {{sitename}}. : is not needed because sitename does not accept any arguments.

Now you may use this magic word in your translations:

*en.json*

"copyright": "{{SITENAME}}. All rights reserved."

*ru.json*

"copyright": "{{SITENAME}}. All rights reserved."

The markup:

*demo.html*

<!-- ... -->

<footer class="translate" data-args="copyright"></footer>

The magic words can be more complex. This one, for example, is going to construct a link with the specified title and URL:

*global.js*

// ...

$.extend($.i18n.parser.emitter, {

sitename: function() {

return "Demo";

},

link: function (nodes) {

return '<a href="' + nodes[1] + '">' + nodes[0] + '</a>';

}

} );

nodes is an array of arguments. Use this magic words just like gender: or plural:

*en.json*

"about": "{{link:About {{SITENAME}}|localhost/about?locale=en}}"

*ru.json*

"about": "{{link:{{SITENAME}}|localhost/about?locale=ru}}"

Note that magic words can be nested, like shown in this example.

**Documenting Your Translations**

Localizing an application [is hard](https://phraseapp.com/blog/posts/12-reasons-software-localization-l10n-dang-hard/). In many cases, simply translating messages [is not enough](https://phraseapp.com/blog/posts/localization-isnt-just-about-translation/) as you need to know their context. Therefore jQuery.i18n's translations can be [documented](https://github.com/wikimedia/jquery.i18n#message-documentation). Usually, it is done inside a file called *qqq.json*. Inside you provide translations keys and their descriptions. For example:

*i18n/qqq.json*

{

"@metadata": {

"authors": [

"Ilya"

]

},

"app-title": "App's title",

"welcome": "Welcoming message. Should be friendly.",

"letter": "Notification about letter from someone. Should be informal.",

"letters": "Total letters count.",

"copyright": "The name of the site (transliterated) and copyrights.",

"about": "Link to About Us page. The URL should contain the proper locale."

}

Having such file in place is really useful.

**Conclusion**

In this article, we've seen jQuery.i18n library by Wikimedia Foundation in action. We discussed all of its main features and hopefully, by now, you feel more confident about using it. Of course, there are other similar (and not so similar) [solutions available](https://phraseapp.com/blog/posts/rails-i18n-setting-and-managing-locales/) so you might want to try them as well.

I thank you for staying with me and see you soon!