Localization - dev best practices

This page is about Localization best practices for developers. If you are looking for information on adding Languages to you deployment go here

Ushahidi is built on Kohana which provides language some localization helpers. See the Kohana docs for more info: http://docs.kohanaphp.com/general/i18n

Tips when creating strings

Never hard code strings

Good

```
<?php echo Kohana::lang('ui_main.approved_reports'); ?>
```

Bad

```
<?php echo 'Approved Reports'; ?>
```

Don't split strings

Example:
You're including a heading 'Approved Reports'.

Good

```
<?php echo Kohana::lang('ui_main.approved_reports'); ?>
```

There's already a translation string for 'Reports' so we could just reuse it:

Bad

```
<?php echo Kohana::lang('ui_main.approved').' '.Kohana::lang('ui_main.reports'); ?>
```

However thats a bad idea..
1. it provides translators with no context
2. it assume the words will always appear in that order, when for example the French translation is more like: 'Rapports approuvés'

Use arguments and string formatting

Example:
You're including a reports count: 'Showing 10 - 20 reports'

Good

```
<?php echo Kohana::lang('ui_main.showing_x_to_y_reports', array($count_min, $count_max)); ?>

// in the ui_main language file
$lang = array(
  'showing_x_to_y_reports' => 'Showing %1$d - %2$d reports',
);
```

Bad

```
<?php echo Kohana::lang('ui_main.showing').$count_min.' - '.$count_max.Kohana::lang('ui_main.reports'); ?>

// in the ui_main language file
$lang = array(
  'showing' => 'Showing',
  'reports' => 'Reports',
);
```

Use explicitly ordered placeholders:

Good

```
'page_of' => 'Page %1$s of %2$s',
  /* Explicit ordering of the replacements,
  even if they are the same order as English */
```

Bad

```
'page_of' => 'Page %s of %s',
  /* Just grabbing the replacements as they
  come and hope they are in the right order */
```

Why? Because ordering my vary in other languages. The 1st version gives translators no context and assumes all languages structure sentences the same way. Using arguments and formatted strings is much better.
In some languages the string transliterated back to English might read something like:

Alternative Text

```
'page_of' => 'Total of %2$s pages, currently on page %1$s',
  /* Explicit ordering of the replacements,
  reversed compared to English as the total comes first */
```

Exception: when there is only 1 placeholder
When theres just 1 placeholder in the string, its ok to use unordered placeholders

```
'ushahidi_release_version' => 'Ushahidi %s',
```

Don't create string in ALL CAPS

Creating strings in all caps means when the design changes and you want search in lower case.. all the translations have to change.
And if themers want the text in lower case, they have to hack the translation files.
If you do this the good way, when the design changes you can just update the code..

Good

```
<?php echo utf8::strtoupper(Kohana::lang('ui_main.search')); ?>

// in the ui_main language file
$lang = array(
  'search' => 'Search',
);
```

Bad

```
<?php echo Kohana::lang('ui_main.search'); ?>

// in the ui_main language file
$lang = array(
  'search' => 'SEARCH',
);
```

Even better might be to replace the string to upper call with a CSS text-transform, making life even easier for themers.

Use utf8 functions when transforming text

PHP's built in string manipulation functions don't always handle UTF8 characters. Use Kohana's utf8 library where you can instead (http://docs.kohanaphp.com/core/utf8)

Good

```
<?php echo utf8::strtoupper(Kohana::lang('ui_main.search')); ?>
```

Bad

```
<?php echo strtoupper(Kohana::lang('ui_main.search')); ?>
```

Related links

- Kohana lang docs http://docs.kohanaphp.com/core/kohana?s[]=lang#use_language_strings
- Kohana internationalization http://docs.kohanaphp.com/general/i18n
- Kohana UTF8 library http://docs.kohanaphp.com/core/utf8