Change Language of html

DOKIMASE TO JSON PROTOU DOKIMASEIS TO APO KATW...(still searching for a way)

Step 1

To start, let’s assume we have the following HTML:

Hello there, how are you?

This DIV layer contains our title. Now, we have decided we want this title to be available in multiple languages. Our first step is adding a class to the div so we can identify it later on:

Hello there, how are you?

Step 2

With that ready, we’re just two steps away. First off, we are going to create an XML file that includes our translation s. In this XML file, we can store translations for multiple phrases and we can easily add more languages at a later sta ge. We shall save this file as languages.xml and save it in the same folder as our HTML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<translations>
   <translation id="title">
      <english>Hello there, how are you?</english>
      <italian>Ciao, come stai?</italian>
   </translation>
</translations>
```

You will store all phrases you want to translate between the <translations></translations> tags. You will store each p hrase in a <translation></translation> tag. In order to identify which phrase is being translated, we need to add the id

=”title”. The name should match the name of the CSS class you assigned in the HTML. Finally, we can put the trans lations inside and surround them by tags defining the language. For instance, we need to put the Italian text in betwe en <italian></italian> tags. Keep in mind that you can easily change the names of these tags – For example, you ma y choose to use <eng></eng> and <ita></ita> instead.

Step 3

With that complete, you just need to add jQuery to read the XML file and replace the contents of your DIVs based o n the language selected. Here you go:

```
<script src="path/to/jquery.min.js"></script>
<script type="text/javascript" language="javascript">

$(function() {
    var language = 'italian';
    $.ajax({
        url: 'language.xml',
        success: function(xml) {
            $(xml).find('translation').each(function(){
                var id = $(this).attr('id');
                var text = $(this).find(language).text();
                $("." + id).html(text);
            });
        }
    });
});
</script>
```

That’s all the code needed – Have a look at it again with comments this time:

```
// Include jQuery script
<script src="path/to/jquery.min.js"></script>

<script type="text/javascript" language="javascript">

//  $(function() { should be used
// each time you use jQuery

$(function() {

    // Here we set the language
    // we want to display:

    var language = 'italian';
```

```
    // In order to get the translations,
    // we must use Ajax to load the XML
    // file and replace the contents
    // of the DIVs that need translating

    $.ajax({

        // Here, we specify the file that
        // contains our translations

        url: 'language.xml',

        // The following code is run when
        // the file is successfully read

        success: function(xml) {

            // jQuery will find all <translation>
            // tags and loop through each of them

            $(xml).find('translation').each(function(){

                // We fetch the id we set in the XML
                // file and set a var 'id'

                var id = $(this).attr('id');

                // This is the most important step.
                // Based on the language we can set,
                // jQuery will search for a matching
                // tag and return the text inside it

                var text = $(this).find(language).text();

                // Last, but not least, we set the
                // contents of the DIV with a
                // class name matching the id in the
                // XML file to the text we just
                // fetched

                $("." + id).html(text);
            });
        }
    });
});

</script>
```

And that’s it! Refresh your page and the Italian version should load, replacing the default English one. In the exampl e above, we set the language manually:

var language = 'italian';

We could just as easily set that via PHP:

var language = '<?php echo $sLanguage; ?>';

Or by reading it from the URL – you can use this jQuery Plugin to do that.
Bonus Trick

As you add more languages, you will realize that phrases are longer in certain languages and shorter in others. We m ight want to have custom CSS for each language. Taking the example above, we would initially have the following:

div.title { font-size:30px; }

What if we wanted the Italian to have a smaller font? Easy! We need to make a slight modification to our jQuery:

```
$(function() {
   var language = 'italian';
   $.ajax({
      url: 'language.xml',
      success: function(xml) {
         $(xml).find('translation').each(function(){
            var id = $(this).attr('id');
            var text = $(this).find(language).text();
            $("." + id).html(text);

            // Here's the new line we're adding.
            // We are assigned the DIV a new class
            // which includes the old class name
            // plus a "_language" - In this case,
            // loading Italian would assign the DIV
            // a "title_italian" class

            $("." + id).addClass(id + '_' + language);
         });
      }
   });
});
```

Now that we've added that line, we can just add the following CSS:

```
div.title { font-size:30px; }
div.title_italian { font-size:20px; }
```

Your Italian text should now be smaller. Note: In order for this to work, you must put the new language CSS definiti ons underneath the default one. Switching those two lines around will not work.
shareimprove this answer
answered Apr 25 '12 at 9:16
Francois
8,78322 gold badges4242 silver badges5454 bronze badges
add a comment
1

Sorry for the late answer, but better late than never... :-)

Francois answer is a good solution for a simple and quick solution.

For a more complete and flexible solution (with plural forms handling, for example...), please have a look at: i18next . They provide:

- support for variables
- support for nesting
- support for context
- support for multiple plural forms
- gettext support
- sprintf supported
- detect language
- graceful translation lookup
- jquery function
- get string or object tree
- get resourcefiles from server
- resource caching in browser
- post missing resources to server
- highly configurable
- custom post processing
- translation ui

I'm using i18next solution myself, though I would personally prefer a server-side solution to avoid any additional bu rden on client side... :-)

N.B. I have no relation at all with i18next.com... :-)