Multi-language Website Generating Python Script

create-language-folders.py allows website application authors to generate separate folders for each language their website is designed to support.
This is done by preparing the following:

- **supported-languages.xml** file that lists the supported languages and their settings.
- **strings.[language key].xml** file for each supported language.
- **strings.xml** file that includes default values, if there is not a falling language, and that allows for creating non-language-specific placeholders in the website (such as page width, etc.)
- **$keys** folder that includes at least all the website-language specific files (normally, all the website files), with placeholders instead of language-specific strings
- optionally, a **create-language-folders.sh** shell script file to run the python script more easily

Usage

python[3.2+] [create-language-folders.py path] [$keys folder's parent folder path]

example:
python3.2 /home/user/workspace/MyWebSite/Root/create-language-folders.py /home/user/workspace/MyWebSite/Root

Operating Systems: linux
Requirements: python3.2 or higher

Dive-Into

create-language-folders.py generates folders along side the **$keys** folder, by the name of the language keys, specified in **supported-languages.xml**, with the corresponding translations read from the **strings.[language key].xml** files. This is done by replacing the placeholders specified in the **$keys** folder files.

The script's scope is defined by file extensions. The default is set for ***.htm, *.php and *.js** but is easily configured in the script header titled **easily-modifyable constants**. Also

defined in this header are the placeholders' start and end marks. By default a placeholder looks like this: **<!$PLACEHOLDER$!>**.

The script can also 'fall' to another language's translation, if a specific translation is not found, and a **falling-language** settings has been defined in **supported-languages.xml** for a given language.

Finally, to support **right-to-left** languages such as Arabic and Hebrew, the python script can also replace <html> tags with <html dir='rtl' > ones, thereby changing the reading order of web pages. If some pages are needed in a specific reading order, setting them in the **$keys** folder with the needed order will exclude them from the python script scope for reading order setting.

Normally, all the required files and folders will reside in one folder. However, since the **$keys** parent folder is provided as a command-line argument for **create-language-folders.py**, the python script only requires that **supported-languages.xml** and the **strings.[language key].xml** files will be located in the same path as its own.

Notes

- **Files with extensions that are not mentioned in the glExtension variable are simply copied as is from the $keys folder to language specific folders (see "easily-modifyable constants header" in this document).**
- **To manage files in each specific language, beyond the scope of the python script, simply omit the files from the $keys folder. This way they will not be overwritten every time the python script executes. This may be necessary for image files that contain language specific texts.**
- **When a value is obtained from the strings.xml it is not HTML escaped into the produced files, as is the case with strings.[language key].xml files. This allows strings.xml to contain some server side code you may wish to parameterize in this way.**

supported-languages.xml structure

The follwing example demonstrates the full settings that can be configured in the **supported-languages.xml** file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<supported-languages>
```

```
<language name="العربية" key="ar" dir="rtl" />
<language name="English" key="en" falling-language="ar" />
</supported-languages>
```

Language **key** is the only **required setting** and designates the folder name for each langauge. It is also used to refer to langauges in the **falling-language** setting. In this example, whenever a string is not found (by its placeholder key) in the strings.en.xml file, it is sought in the strings.ar.xml file.

Language **name** is not used by the script and only serves the website author to easily and unequivocally identify each supported languages.

Language **dir** is set for languages that require a different html dir element (setting alignment and reading order of the page). Normally, it is used for **Right-to-Left** languages, such as Arabic and Hebrew.

XML **encoding** has to be always UTF-8.

strings.[language key].xml and strings.xml structure

The follwing example demonstrates the strings.[langauge key].xml or strings.xml structure:

file name: **strings.en.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<strings>
<str>
    <key>ACCESS_DENIED</key>
    <value>Access Denied</value>
</str>
<str>
    <key>LOGIN_INCORRECT</key>
    <value>Incorrect User name or password.</value>
</str>
<str>
    <key>LOGIN_NAME_REQUIRED</key>
    <value>User name is required.</value>
</str>
<str>
```

```
    <key>PASSWORD_REQUIRED</key>
    <value>Password is required.</value>
</str>
</strings>
```

The **language key** in the strings file name determines to which language does this translations apply. In this case it is English (language key = en).
Each **key** element contains the placeholder key, without the starting and ending marks that will be found in the **$keys** folder files.
Each **value** element contains the language's translated text.

XML **encoding** has to be always UTF-8.

Placeholders file example

The following examples demonstrate how a placeholders containing files might look like:

1) file: **$keys\AccessDenied.htm**

```
<!DOCTYPE HTML>
<html>
<body>
<h1><!$ACCESS_DENIED$!></h1>
</body>
</html>
```

The placeholder **<!$ACCESS_DENIED$!>** includes the **<!$** starting mark and **$!>** ending mark.

2) file: **$keys\login.php**

```
<?php

if ($_SERVER[ 'REQUEST_METHOD'] == 'POST')
{
   ...
  if ( $nLoginFlags == 0 )
   {
```

```
        header( 'Location: ' . Consts::URL_HOME );
        exit();
    }
    else
    {
        switch(  $nLoginFlags )
        {
            case Login::ERR_LOGIN_INCORRECT_NAME_PASSWORD:
                $sMessage .= '<!$LOGIN_INCORRECT$!><br/>';
                break;
            default:
                if ( $nLoginFlags & Login::ERR_LOGIN_NAME_EMPTY ==
Login::ERR_LOGIN_NAME_EMPTY )
                    $sMessage .= '<!$LOGIN_NAME_REQUIRED$!><br/>';
                if ( $nLoginFlags & Login::ERR_LOGIN_PASSWORD_EMPTY ==
Login::ERR_LOGIN_PASSWORD_EMPTY)
                    $sMessage .= '<!$PASSWORD_REQUIRED$!><br/>';
                break;
        }
    }
}
?>
<!DOCTYPE HTML>
<html>
…
```

Placeholders can be inside **php**, **js**, etc. segments as well.

easily-modifyable constants header

The header **easily-modifyable constants** within **create-language-folders.py** contains a number of variables that can be easily modified to meet your requirements:

```
#---------------------------------------------------- easily-modifyable constants
```

```
# the file extensions to distinguise while processing the $keys folder
glExtensions = [ 'htm', 'js', 'php', 'css' ]
# whether files having the above extensions contain placeholders that should be processed (=True) or just copied as is to the language-specific folder (=False)
gbProcessMatchingFiles = True
# placeholder starting mark
gsPHStart = '<!$'
# placeholder ending mark
gsPHEnd = '$!>'

# the file extensions to set the html dir element in, if the langauge requires (such as in Arabic and Hebrew)
glDirElementExtensions = [ 'htm','php' ]
# HTML tag to replace. Only html tags written exactly like this, will be replaced by <html dir='rtl'> when required (in Arabic and Hebrew, for instance)
gsHtmlTagForDirElement = '<html>'
# HTML tag count to replace by <html dir='rtl'> when required (in Arabic and Hebrew, for instance). 1 means only the first <html> tag will be replaced
gnHtmlTagCountForDirElement = 1
```

create-language-folders.sh

The shell script file **create-language-folders.sh** simply contains the command-line to run the python script. Since website authoring is an ongoing task, such a script helps regenerating the language specific folders without the need to open a command-line

window.

This is an example of how the **create-language-folders.sh** might look like:

```
python3.2 /home/user/workspace/MyWebSite/Root/create-language-folders.py
/home/user/workspace/MyWebSite/Root
```

Note: To make **create-language-folders.sh** executable, run this line, for instance, in the command line:

```
chmod 755 [create-language-folders.sh path]
```