WLED Localization

Localization for WLED provides the ability to switch between display languages dynamically from the UI settings page. This setting is local to the browser, and is effective immediately (no Save or page refresh is required).

The rest of this document explains the process for providing this capability.

In summary, the UISimulator Web Server and a couple of python scripts are used to extract, translate, and test the translations on a static simulation of WLED. The translation files are then committed to Github, from which the L12N script included on each HTML page will access and apply the translation.

There are 4 subprocesses with supporting tools, detailed below.

1. I18N: Extracting text displayed in the WLED UI by 'screen scraping'. Text is extracted from text nodes and certain attributes. The UI simulator is used to perform the screen scraping.
   It is possible to manually change some of the scraped text into regular expressions.
   In addition, data from JSON end-points can be manually saved from a running instance to complement the screen scraping (currently /json/fxdata only is recommended).

2. Translation into desired languages indicated by their ISO639-1 codes, using the Google Translation API. The resulting translation file lists the phrases and where they occur. The translation text can be manually adjusted. If a phrase requires different translations depending on its context, the entry can be manually changed to contain multiple translations with the associated context.
   Before these translation files can be used, an optimization script must be applied to remove extraneous context information and to output the optimized format used for testing and deployment.

3. L12N: applying the translated text for validation to the translated UI using the provided UISimulator Web Server.

4. Deployment of the translated UI, comprising: a. committing the phrase translation file for each language to Github b. for each HTML page inserting the L12N.js script and calls to the setLang() function whenever the UI is changed. This is a one time change. A script is provided to make these changes. c. Building of the UI and application with the standard tools.

UI Simulator

The UI Simulator is a python Web Server which serves up a static emulation of the WLED UI. It also injects scripts used for the I18N and L12N processes, and accepts POST back of scraped text. One of three modes can given as a parameter when launching the Simulator:

1. (No mode). The HTML text is served up without changes.

2. I18N. Into every HTML page it serves, the Simulator injects the I18N script (or replaces the L12N script if present), and adds a call to run18N in body.onload() to do the screen scraping. The I18N.js script is also used to wrap functions such as genForm and showToast, that add text in the UI, to capture that text. (Equivalent changes can also be done directly into the pages if desired).
   At the end of a screen scraping invocation, the captured text is POSTed back to the Simulator, which

saves it. The screen scraping functions are incremental, ignoring previously reported text.
If desired, screen scraping can be manually invoked as **I18N.singleton.scrape(optionalTag)** from the browser developer tools.

3. L12N. This is similar to the I18N mode, except the L12N.js script is injected. As a consequence, instead of text capture a selected translation is applied. This is useful in checking translations.

Process tools

Process	Tool	Inputs	Outputs
I18N	UISimulator	UI. The user exercises the UI as much as possible in order to capture as much text as possible.	I18N/data/*
Translation	L12Nv2.py	I18N/data/*	I18N/L12N/XX.json
(Translation file finalization)	installLangs.js	I18N/L12N/<la>.json I18N/data/iso639.json	I18N/langs/<la>.json
Deployment	injectL12N.py	wled00/data/* or I18N/list.json	wled00/data/* File formatting is preserved.

The output files from the I18N and Translation (L12N.py) can be manually adjusted, and those changes will be preserved across invocations of the various tools.

Manual adjustments

Adjusting I18N/fromPage/* files

Here is a standard entry.

```
{
   "content": "Segment 0"
},
```

We can change this exact phrase to use a pattern, as follows:

```
{
   "content": "Segment 0"
   "pattern": "^Segment ([0-9]+)$"
},
```

This will be translated to L12N/.json to:

```
{  "exact: { ... }
   "pattern": { ...
        "Segment 0": {
            "text": "Segment 0",
            "translations": [
                {
                    "translation": "セグメント $1",
                    "for": {
                        "_settings_ui.json": [ 234 ],
                        "index.htm.json": [ 512 ]
                     }
                }
            ],
            "pattern": "^Segment ([0-9]+)$"
        }
```

Adjusting L12/*.json files

In the above example, we can see the translations object is an length 1 array of translation text (in this case a regular expression replace string) and where that translation applies. If the translation must be different depending on the context, the single entry can be split up.

```
{
          "translations": [ translation1:{for :{...}}, translation2:{for :
{...}}]
},
```

Required Libraries

The scripts use python3 and the libraries beautifulsoup4, googletrans (I used 3.1.0a0), and a 1-line patched version of HTMLParser.

Future work

The I18N process might be done more effectively using an external screen scraper working on a running instance of WLED.