Proposal for Google Summer of Code (GSoC) 2025

Project Title: Internationalization with AI Translation Support for Music Blocks

Organization: Sugar Labs

Personal Information

Name: Aman Chadha
GitHub Profile: AmanChadha
GitHub PR for Music Blocks: PR #4459

Already Merged PR in Music blocks:

#4437: Fixed Tests for Turtle Singer and Synth Utils

#4433: Fix PO File Errors

Other Merged PR:

#5523: Manipulation: Make jQuery.cleanData not skip elements during cleanup(JQuery Library).

About Me

I'm a passionate open-source contributor with a strong focus on AI and music technology. I actively engage on platforms like **Stack Overflow (reputation: 2496)** and have contributed to multiple open-source projects, including Sugar Labs. My merged PRs in Music Blocks, such as fixing Turtle Singer tests (#4437) and resolving PO file errors (#4433), showcase my ability to tackle technical challenges. I thrive at the intersection of creativity and innovation, blending cutting-edge neural networks with music technology to create meaningful solutions.

Synopsis

Music Blocks currently uses an outdated internationalization (i18n) system called webL10n.js, which lacks support for modern i18n features such as pluralization and language-specific formatting. This project aims to modernize the i18n system by migrating to a contemporary JavaScript i18n framework (specifically i18next) and integrating AI translation services to assist human translators. By leveraging tools like Google Translate or DeepL, initial translations can be generated automatically, reducing manual effort and supporting additional languages more efficiently.

Benefits to the Community

- Enables more accurate and culturally appropriate translations with modern formatting capabilities.
- Reduces the workload on human translators by providing automatic initial translations.
- Supports more languages, enhancing accessibility and inclusivity.
- Facilitates a streamlined translation process for future updates.

Deliverables

1. Migration from WebL10n to i18next
2. Integration of AI Translation Support
3. Creation and Update of JSON Translation Files
4. Documentation for Future Maintenance
5. Automated Testing for Translation Accuracy

Technical Details

1. Migration from WebL10n to i18next

The migration involves replacing the outdated webL10n.js system with the modern i18next framework. This change affects several key files:

- ``The function _() is updated to use i18next.t() for translation lookups. Key features added include:
    - Fallback strategies for translation:
        - Cleaned text (without special characters)
        - Lowercase version
        - Title case version
        - Hyphenated text
    - Support for specific languages like Kanji and Kana with sub-language handling.
    - Pseudo code version of _() function which would be used for this translation procedure. You can check the implementation in my PR #4459.

```
Function _(text, options={}):
    If text is empty → return ""

    Clean text by removing special characters
    Set language and check if it's "kanji" or "kana"
        If yes, try fetching translation with special suffix
        If not found, fallback to normal translation

    Try translating text in different variations:
        1. Original text
        2. Cleaned text (without special characters)
        3. Lowercase version
        4. Title-case version
        5. Hyphenated version (spaces → "-")

    If translation is still missing, return original text

    Preserve original text's capitalization:
        - If all uppercase → return translated text in uppercase
        - If all lowercase → return translated text in lowercase
        - If title-case → return translated text in title-case

    Return final translated text
```

- WebL10n.js script tags are removed, and i18next is imported.

2. Improving String Handling & RTL Support in i18n Migration

As part of the migration from WebL10n to i18next, we need to address certain string-handling limitations and ensure better support for Right-to-Left (RTL) languages.

2.1 Resolving String Concatenation Issues

In the current codebase, some strings are concatenated like:

const name = currentKey + " " + _(mode);

This approach does not work well for languages with different word orders (e.g., RTL languages). Instead of manually concatenating strings, we will use **interpolation with placeholders**:

const name = _("music.mode", { key: currentKey, mode: mode });

for eg: In English: "C Major"

In French: "Do Majeur" (Order changes)

In Arabic (RTL): كبير C (Word order reversed)

In my _() function I have added new optional parameter for the same you can check in utils/utils.js

3. Optimizing Translation Keys for Maintainability

- Instead of using full sentences as keys, we should use short, descriptive keys. For Eg.
  {
  "The Semi-tone transposition block will shift the pitches contained inside Note blocks up (or down) by half steps.": "El bloque de transposición de semitonos desplazará los tonos dentro de los bloques de nota hacia arriba (o abajo) en semitonos."
  }

  should change to

  {
  "block.semitone_transpose": "El bloque de transposición de semitonos desplazará los tonos dentro de los bloques de nota hacia arriba (o abajo) en semitonos."
  }
  Making it easier for developers and translators to manage.

4. Unified JSON Structure for Japanese

- Previously, Kana and Kanji translations were stored in separate JSON files.
- Now, they have been merged into a single JSON file under locales/ja.json.
- This simplifies maintenance and avoids redundant translation efforts.
- When the user selects Japanese (Kana) or Japanese (Kanji), the application correctly applies the preferred writing system.

For example, earlier we had two separate files**: ja.json** and **ja-kana.json**. Instead of maintaining two separate files, we will now consolidate them into a single file called **ja.json**. This file will contain subkeys for "kana" and "kanji," each holding the corresponding translations of the relevant words, as shown below

```
{
  "Refresh your browser to change your language preference.": {
    "kanji": "言語を変えるには、ブラウザをこうしんしてください。",
    "kana": "げんごを かえるには、ブラウザを こうしんしてください。"
  },
  "project undefined": {
    "kanji": "プロジェクト未定義",
    "kana": "プロジェクトみていぎ"
  },
  "action": {
    "kanji": "アクション",
    "kana": "アクション"
  },
  "duck": {
    "kanji": "あひる",
    "kana": "あひる"
  }
}
```

5. i18next Language Initialization & Loading

- The loader.js file initializes i18next, loads the translations from **locales/{{lng}}.json** and updates the UI dynamically.
- The key Separator: '=)' is used to handle structured keys correctly.

6. Handling Language Preferences in

- If a user has previously selected a language, it is stored in **this.storage.languagePreference**.
- For Japanese:
    - If ja is selected, it appends the Kana preference (ja-kana or ja-kanji).
    - This ensures the correct script is used based on user preferences.

7. Language Selection Logic in Language Selector Dropdown

- When a user selects a language from the dropdown:
    - The preference is stored.
    - For Japanese, ja-kana or ja-kanji is explicitly set.
    - A message prompts the user to refresh the browser to apply changes.

8. Translation Key Lookup with Fallback (``)

- If a key does not exist in the selected language, it falls back to more regionally or linguistically similar languages. This way, the fallback would feel more natural to users.

For instance:

- Languages like Quechua, Aymara, and Guarani could fall back to Spanish since it is more familiar in those regions.
- Bengali, Nepali, and Punjabi could fall back to Hindi, which is widely understood in South Asia.
- Haitian Creole, Wolof, and Lingala could fall back to French due to historical and linguistic ties.
- Brazilian and Angolan Portuguese could fall back to European Portuguese.
- Somali, Hausa, Pashto, and Persian could fall back to Arabic as it shares more linguistic commonality.
- Simplified and Traditional Chinese could fall back to their respective Chinese variants.
- This ensures that missing translations do not break the UI.

9. AI Translation Support with Context Awareness

To streamline and enhance the translation process, a Python-based script translate_ai.py has been developed. This script leverages services like the Google Translate API to automatically fill in missing phrases within JSON translation files.

Key Capabilities:

- Generates new JSON files for additional languages.

- Updates existing translations, identifying and filling gaps automatically.
- Logs all translation updates for manual review by translators.

Context-Aware Translation

A key enhancement is the use of an ***en_context.json*** file, which stores additional semantic context for each English string. This allows the AI to:

- Understand how each phrase is used in the UI or app (e.g., button label, instructional text, noun vs. verb).
- Guide the translation model for more accurate, domain-appropriate results.

Example: Word “duck”

- Without context: The word *"duck"* could be translated as either a bird or an action.
- With context: If the context is "General UI string" or "animal name", the translation engine leans toward the correct noun form, ensuring accurate localization.

This intelligent integration of context reduces ambiguity, increases consistency across languages, and minimizes the need for post-editing. It significantly improves the quality of automatic translations, especially in cases where short or polysemous terms appear.

10. Translation Accuracy Verification (Enhanced with Contextual Reference)

To ensure high-quality and semantically correct translations across all supported languages, we implemented a two-step verification strategy:

Back-Translation with Context-Aware Evaluation

We perform back-translation, where the translated string is translated back into English. This back-translated version is then compared with the original English string.
What makes our approach robust is the use of the **en_context.json** file, which contains not only the source strings but also the semantic context (e.g., "A complete sentence", "General UI string"). This context ensures that even if back-translation returns a slightly paraphrased version, we can assess whether the meaning is preserved, not just the wording.

Example from the Catalan translation file:

- **Original:** *"Refresh your browser to change your language preference."*
- **Translated (ca):** *"Actualitzeu el navegador per canviar la vostra preferència d’idioma."*
- **Back-Translated:** *"Update your browser to change your language preference."*
- **Context:** *"A complete sentence, likely a message or instruction"*

Here, the wording is different, but the meaning remains intact — due to context-aware analysis.

Lexical Similarity Scoring

To support this qualitative analysis with a quantitative measure, we compute the lexical similarity between the original and the back-translated string using the **SequenceMatcher** ratio. This metric gives a percentage indicating how closely the two strings align.

Example:

- Original: *"Refresh your browser to change your language preference."*
- Back-Translated: *"Update your browser to change your language preference."*
- Lexical Similarity Score: 90.09%

Threshold and Logging

If the similarity score drops below 60%, the translation is flagged for manual review. This score, along with the back-translation, context, and flags, is saved in a structured JSON report for further analysis. This system provides clear visibility into which translations may need human intervention, especially for nuanced or ambiguous strings.

I have attached the report.json in the link given below:

https://github.com/sugarlabs/musicblocks/pull/4459#issuecomment-2779421450

11. Future-Proofing Translations

To maintain consistency over time:

- All new phrases must first be added to **en_context.json**, including both the text and a clear context description.
- Scripts will detect new entries and ensure they are translated and appended only in other language files, without overwriting existing data.
- This process ensures version control, minimizes rework, and maintains a consistent context-driven translation flow across updates.

12. Changes to PO Files

- Since the JSON structure has changed, PO files may need updates to align with the new format.
- Initially, we will convert all existing .po files to .json format to maintain compatibility. However, moving forward, newly translated files will be created directly in .json format using our AI translation code. This approach simplifies the process by eliminating the need for .po files in the future.
- Additionally, this transition eliminates the need for the **.ini** file previously used for configuration, as i18next handles this internally.
- Specifically, we need to decide:
    - Should untranslated Kana default to Kanji or English?

Project Timeline

Period	Task
May 20 - June 10	Research i18next features, finalize migration plan
June 11 - July 5	Migrate key files to i18next, replace webL10n
July 6 - July 20	Implement AI translation support (translate_ai.py)
July 21 - Aug 10	Generate and validate translation files
Aug 11 - Aug 15	Implement automated testing for translations
Aug 16 - Aug 20	Documentation and final testing

Expected Outcomes

- A modern i18n system in Music Blocks using i18next.
- A functional Python script for AI-assisted translations.
- Automated tests to ensure translation accuracy.
- Comprehensive documentation for future developers.

To Get Involved

- Review my PR on GitHub: PR #4459
- Test the AI translation tool with various languages.
- Suggest additional features or improvements.