Support for Multilevel Subdivisions, Multi-Subdivision Queries, and Localization System Revamp

Google Summer of Code 2025 Proposal

Open World Holidays Framework

(Python Software Foundation)

ABSTRACT

This project aims to significantly improve the Holidays Framework's regional and localization capabilities by implementing three major features:

1. Add multi-level subdivision support (#1713)

2. Extending the `subdiv` argument to accept multiple subdivisions (e.g., lists, sets, tuples) for flexible queries involving multiple subdivisions (#1445)

3. Overhauling the localization system to generate `.po` files on per-locale basis. (#1658)

These changes will make it easier for users to work with holidays from multiple regions, simplify the translation process, and ensure the framework remains scalable and easy to maintain for over 166 entities. The project focuses on the internal logic refactoring, validation, unit testing, edge case handling, and finally updating the documentation while keeping it backward compatible.

ABOUT ME

Name: Roshan Pradhan

Email: roshanpradhan2048@gmail.com

Github: https://github.com/Roshan-1024

Linkedin: https://www.linkedin.com/in/roshan-pradhan-95911b341/

LeetCode: https://leetcode.com/u/Roshan2048/

Time Zone: IST (UTC +5:30)

Academic Background:

- I am a first year student from IIIT Kalyani pursuing B.Tech in Computer Science Engineering.
- I started programming in Python when I was in high school and I have experience working with other programming languages like C, C++, JS.
- I found out about Holidays Framework through an Open Source Event named Winter Of Code 4.0 conducted by our college where I did my first open source contribution. I have experience working with Holidays Framework since 4 months and I have a decent understanding of the codebase.

Achievements:

- Secured 3rd place among 5100+ registrations in Winter of Code 4.0
- Practiced more than 200 LeetCode problems
- Completed the Machine Learning Specialization by Andrew Ng.

Motivation:

I am extremely excited about this project as it is a combination of technical challenges and real-world impact. And having contributed to Holidays Framework before I’m motivated to take on a larger and more impactful project.

What do I expect from this project and from my mentors?

From my mentors, I would really appreciate their insights, getting feedback on my approach and advice on writing clean and efficient code.

Other Commitments During GSoC:

During the summer, I have no other obligations. Following the conclusion of my vacation, there will be a relatively light workload after classes resume from the second week of July, allowing me to commit approximately 25-30 hours per week to the project on average. Should any circumstances arise that impede the project's progress, I will promptly inform my mentors and make up for it by increasing my workload. In the event of unforeseen obstacles, I am prepared to allocate additional time to the project in the subsequent weeks.

Post-GSoC commitments:

After GSoC, I will continue contributing to Holidays Framework specifically adding support for holidays for other countries. I also want to stay engaged with the community and help other contributors.

Open and Merged PRs:

(Merged) #2289: Migrate Documentation from Sphinx to MkDocs

(Open) #2402: Add Trinidad and Tobago Holidays

Multilevel Subdivision Support for Holidays Framework (#1713)

Synopsis:

Extend the Holidays Framework to support multilevel subdivisions beyond the existing L1 (State) model. This enhancement will introduce a flexible and scalable system capable of handling nested administrative divisions like L2 (District), L3 (City), and beyond, preserving compatibility with the current system and improving global localization accuracy.

Benefits to the Community:

• Hierarchical Accuracy: Users can define holidays more precisely based on deep administrative levels.
• Backward Compatibility: Legacy L1 behavior remains unaffected.
• Improved Usability: Consistent logic for subdivision inheritance and fallback.
• Better Internationalization: Enables support for complex geopolitical structures (e.g., India, Russia, Germany).
• Scalability: Future-proof design allows easy integration of new nested structures without architecture overhaul.

Deliverables:

• Subdivision mapping structure supporting L0 (Country) → L1 → L2 → L3...
• Upward traversal logic to collect holidays from nested to root level.
• Backward-compatible integration with the legacy holiday generation system.
• Updated subdivision data with nested structure for at least one country.
• Tests verifying nested holiday resolution logic.
• Documentation for contributors on defining and using nested subdivisions.

Step-by-Step Implementation Plan:

1. **Design Phase**
   ◦ Finalize subdivision hierarchy model and data format (eg. dictionary).
   ◦ Define mapping convention: each subdivision maps to its direct parent subdivision.
2. **Parent Mapping Logic**
   ◦ Implement `SUBDIVISION_PARENT_MAP` where each subdivision code maps to its parent subdivision code.
   ◦ Set `None` or country code for top-level subdivisions.
3. **Traversal & Merge System**
   ◦ For a given L2+ subdivision, collect holidays defined in its scope.
   ◦ Traverse upward using the mapping and accumulate holidays from all parent subdivisions.
   ◦ Merge collected holiday sets in order.
4. **Legacy Integration**
   ◦ Modify internal holiday resolution to accept merged holiday list.
   ◦ Ensure that if only L1 is passed, behavior remains unchanged.
5. **Conflict Resolution**
   ◦ Define key naming conventions to avoid duplication or ambiguity.
6. **Subdivision Data Refactor**
   ◦ Pick a test country and structure its subdivisions hierarchically.
   ◦ Update its data to reflect L1 → L2 relationships.
7. **Testing**
   ◦ Write unit tests to validate multi-level traversal logic.
   ◦ Test L1, L2, and mixed inputs for correctness.
8. **Documentation**
   ◦ Provide updated developer guide with hierarchy structure and usage examples.
9. **Pull Request and Migration Summary**
   ◦ Submit PR with thorough migration notes.
   ◦ Describe the backward compatibility and test coverage.
   ◦ List some countries prepped for nested subdivisions as examples.

Extend `subdiv` Argument to Accept Multiple Values (#1445)

Synopsis:

This project proposes extending the `subdiv` argument in the Holidays Framework to accept multiple values (e.g., `list`, `set`, `tuple`), thereby allowing users to combine holidays from several subdivisions within a country. This will greatly increase flexibility and real-world applicability, especially for users needing regional holiday aggregations.

Benefits to the Community:

- **Improves usability:** Supports more realistic use cases by allowing multi-region queries.
- **Easier integration:** Simplifies aggregation of holiday data for tools like scheduling platforms.
- **Simplified user logic:** Removes the need for manual combination of regional holidays.

Deliverables:

- Support for `subdiv` to accept an **iterable** of subdivisions (`list`, `set`, `tuple`), while maintaining backward compatibility with **string** inputs.
- Validation logic for multiple `subdiv` codes, ensuring all are valid.
- Fallback to legacy behavior when a **single string** is passed.
- Refactor of internal holiday computation logic to aggregate results from multiple subdivisions.
- Extensive unit test coverage for multi-`subdiv` logic
- **Edge case handling:**
    - Duplicated or invalid subdivision codes.
    - Invalid Input Type

    - Overlapping holidays with the same name.
- Consistent behavior when `subdiv="ALL"` is used.
- Documentation updates with usage examples for the new feature.

Step-by-Step Implementation Plan:

1. **Input Type Validation and Normalization**
    - Detect if `subdiv` is a **string** or an **iterable** (`list`/`tuple`/`set`).
    - Convert iterable into a `list` for processing.
2. **Fallback to Legacy Behavior**
    - If a **single string** is provided (legacy case), preserve existing logic unchanged.
    - This guarantees backward compatibility for existing users.
3. **Validation of Subdivision Codes**
    - Verify each `subdiv` code against the list of valid codes for the specified country.
4. **Aggregation Logic**
    - Refactor internal logic to iterate over all valid subdivisions.
    - Merge holiday data from each subdivision, avoiding duplicates.
    - For `subdiv="ALL"`, retrieve holidays for **all** subdivisions as a special case.
5. **Unit Testing**
    - Add tests for **single-subdivision** input (legacy behavior).
    - Add tests for **multiple subdivisions**, including overlapping and non-overlapping scenarios.
    - Include tests for invalid or duplicate codes.
    - Cover countries with no subdivisions as well.
    - Test `subdiv="ALL".`
6. **Edge Case Handling**
    - Address complex or nested subdivisions.

- Manage multiple holidays falling on the same date.

7. **Documentation**
    - Update `README.md` explaining the usage of the new feature.
    - Provide clear code examples demonstrating how to pass multiple subdivisions.
    - Clarify behavior of `subdiv="ALL"` using examples.

Generate l10n File on per locale basis (#1658)

Synopsis:

Consolidate all entity-based .po files into a single .po file on a per locale basis. This transformation will streamline the localization structure, minimize maintenance complexity, and provide a scalable solution for supporting translations across 166+ entities.

Benefits to the Community:

• **Improved Translation Workflow:** Easier to manage and update the translation of a single file per language.
• **Cleaner Repository Structure:** Reduced number of files and directories under `holidays/locale/`.
• **Compatibility with Translation Platforms:** Better integration with Weblate.
• **Maintainability:** Easier testing and a clearer overview of translation coverage.
• **Scalability:** A simplified model that scales well with the addition of new locales.

Deliverables:

• A restructured localization system that generates one `.po` and `.mo` file per locale.
• Migration logic to preserve existing translations in the new format, accounting for naming collisions.
• Updated `generate_po_files.py` and `generate_mo_files.py` scripts.
• Clear documentation for translators and developers on the new workflow.
• Testing to ensure translations remain intact.
• Naming convention logic (based on discussion with mentors) to prevent key overlap.
• Compatibility to ensure the existing Holidays Framework codebase can load translations correctly.

Step-by-Step Implementation Plan:

1. **Backup**
   ◦ Backup all existing `.po` and `.mo` files from `holidays/locale/`.
2. **Analysis Phase**
   ◦ Identify all unique locales currently used.
3. **Update POT Generation Logic**
   ◦ Modify the existing logic to generate portable object template (`.pot`) files per entity as an intermediate step.
   ◦ Instead of saving `.po` files per entity, merge their contents into one `.po` file per locale.
4. **Handle Overlapping Keys**
   ◦ Follow mentor-suggested conventions to distinguish conflicting keys.
5. **PO File Creation and Merging**
   ◦ For each locale, create a single `.po` file.
   ◦ Merge all relevant `.pot` data into it.
6. **Translation Mapping**
   ◦ Use backed-up `.po` files to plug in existing translations.
   ◦ Cross-check for untranslated strings and mark them for future translation.
7. **MO File Generation**
   ◦ Refactor `generate_mo_files.py` to compile the new `.po` files into corresponding `.mo` files per locale.
8. **Testing**
   ◦ Ensure the new localization files load correctly.
   ◦ Validate that holiday names are accurately translated in all locales.
9. **Documentation**
   ◦ Create a separate translator guide.
   ◦ Provide examples of the new `.po` file structure and translation process, including the conventions used.
10. **Pull Request and Migration Summary**
    ◦ Submit PR with detailed summary of changes.

Size in Hours: 350 hours

Detailed Timeline

● **Community Bonding Period** (May 8 - June 1)

During this period, I will coordinate with the mentors to finalize the implementation details, validate assumptions, and gather feedback on the proposed plans. I'll familiarize myself deeply with the codebase related to localization and subdivision handling. If discussions conclude early, I will begin preparatory coding work for multi-level subdivision support.

● **Phase 1** (June 2 - June 23, About 3 weeks)

Focus: Add multi-level subdivision support (#1713)

1. Design and implement `SUBDIVISION_PARENT_MAP` for hierarchy.
2. Create traversal system to merge holidays up the subdivision chain.
3. Update internal holiday resolution to support hierarchy-aware merging.
4. Refactor test country (e.g., India) to follow L1 → L2 model.
5. Write unit tests for hierarchy logic and ensure backward compatibility.
6. Document new hierarchy model and usage examples.

● **Phase 2** (June 24 - July 21, About 4 weeks)

Focus: Implement support for multiple values in `subdiv` (#1445)

1. Implement input normalization and validation logic.
2. Ensure legacy behavior is preserved and backward compatibility is maintained.
3. Refactor holiday aggregation logic to handle iterables.
4. Add support for `subdiv="ALL"`.
5. Implement extensive unit tests covering edge cases, duplicates, invalid codes, and countries without subdivisions.
6. Update documentation and examples in README.md.

● **Phase 3** (July 22 - August 18, About 4 weeks)

Focus: Overhaul localization file structure (#1658)

1. Backup and analyze current `.po`/`.mo` files.
2. Modify `.pot` generation to output per-entity templates.

3. Create merged `.po` files per locale.
4. Handle overlapping/conflicting keys via naming conventions.
5. Use backups to migrate existing translations.
6. Refactor `generate_mo_files.py` for new structure.
7. Test loading of new `.mo` files and verify accuracy.
8. Write a dedicated translator guide for new system.

● **Final Week** (August 19 - August 25)

This period will be focused on polishing the implementation across all three features. I'll ensure documentation is finalized, clean up code as per mentor review, and get the PRs ready for merge.

● **Buffer Week** (August 26 - September 1)

Used for any unfinished tasks from previous phases or final refinements. If all primary tasks are completed, this week will be used to explore extended goals such as adapting more countries for nested subdivisions.