

# format\_time Developer Guide

---

## Environment Setup

Python 3.12 Required: The format\_time system is built for Python 3.12. Install Python 3.12 or later and create a virtual environment:

```
python3.12 -m venv venv
source venv/bin/activate
```

Dependencies: Install the core dependencies:

```
pip install numpy SQLAlchemy
```

Optional extras: • Geo: rasterio, scipy, geopandas, shapely, pyproj • UI: PyQt5, matplotlib • Dev: black, isort, flake8, pre-commit, pytest

For all extras:

```
pip install format_time[geo,ui,dev]
```

Installation: From PyPI:

```
pip install format_time
```

Or from source:

```
git clone https://github.com/ProphetGang/format_time.git
cd format_time
pip install -e .
```

CLI Tool: After installation, run:

```
format_time --help
```

Database Configuration: To specify a custom SQLite path in code:

```
from parameters.manager import ParametersManager
pm = ParametersManager(db_url="sqlite:///path/to/my_params.db")
```

---

## Architecture Overview

format\_time consists of: • Time Engine (Time): Custom calendar management. • SunCalendar: 2D sun altitude table generation. • MoonCalendar: 2D moon altitude, phase, brightness. • SunCycleAPI / MoonCycleAPI: Safe runtime queries. • UnifiedTimeModule: Orchestrates components and auto-updates. • ParametersManager: Configuration storage and notifications. • NotificationManager: In-process pub/sub. Data flow: parameters → calendar generation → APIs; selective regeneration on changes.

---

## Module-Level Documentation

time\_engine.time Class: Time Constructor: Time(ticks\_per\_hour: int = 1, hours\_per\_day: int = 24, days\_per\_month: int = 30, months\_per\_year: int = 12, lunar\_cycle\_days: int | None = None) Methods: • advance(ticks: int) • current\_datetime() -> dict • set\_datetime(year, month, day, hour, tick)

---

### time\_engine.sun\_calendar

Function: atomic\_save\_sun\_calendar(array, path) Class: SunCalendar(time: Time, latitude: float = 0.0) Methods: • generate() -> np.ndarray • save(path), load(path), ensure(path) • altitude(day, tick) -> float, clamped to [0,90] • zenith(day, tick) -> float

---

### time\_engine.moon\_calendar

Function: atomic\_save\_moon\_calendar(array, path) Class: MoonCalendar(time: Time) Methods: • generate() -> np.ndarray • save(path), ensure(path) • altitude(day, tick) -> float • phase\_fraction(day) -> float • brightness(day) -> float

---

### time\_engine.api

Class: SunCycleAPI(sun\_table: np.ndarray) • altitude(day, tick), zenith(day, tick) Class: MoonCycleAPI(moon\_table: np.ndarray, lunar\_cycle\_days: int) • altitude(day, tick), phase(day, tick=0), zenith(day, tick)

---

### time\_engine.unified\_time\_module

Class: UnifiedTimeModule(data\_dir: str = None, params: ParametersManager = None) Behavior: • Sets up data directories and file paths. • Initializes ParametersManager and subscriptions. • Generates/loads calendars; exposes sun\_table, moon\_table, sun\_api, moon\_api. Methods: • \_on\_param\_change(key, value) • rebuild\_calendars() • \_ensure\_calendars()

---

### parameters.manager

Class: ParametersManager(db\_url: str = ..., notifier=None) Methods: • get(group, key, default=None) • set(group, key, value) • on\_change(group, key, callback) • get\_all() • reset()

---

### time\_engine.notification

Class: NotificationManager() Methods: • subscribe(key, callback) • notify(key, value) • stop() • reset()

---

### time\_engine.clock

Function: main() CLI entry point (format\_time script) with flags for advancing time, generating calendars, and queries.

---

## API Usage Guide

Time Usage: `from time_engine.time import Time t = Time(...) t.advance(...) ...`

Sun/Moon Calendars: `from time_engine.sun_calendar import SunCalendar from time_engine.moon_calendar import MoonCalendar ...`

Runtime APIs: `from time_engine.api import SunCycleAPI, MoonCycleAPI ...`

UnifiedTimeModule: `from parameters.manager import ParametersManager from time_engine.unified_time_module import UnifiedTimeModule ...`

Parameters & Notifications: `from parameters.manager import ParametersManager from time_engine.notification import NotificationManager ...`

---

## Extending and Customizing

- Custom Time Configs via Time or ParametersManager.
- Adjust `lunar_cycle_days` for moon cycles.
- Subclass SunCalendar/MoonCalendar for custom models.
- Subscribe to parameter events with `on_change`.
- Specify `data_dir` in UnifiedTimeModule.
- Add tests in `tests/` using existing patterns.

---

## Test Suite Summary

- Time Tests: rollover logic, validation.
- Calendar Tests: shapes, ranges, atomic saves.
- API Tests: index wrapping and lookup correctness.
- ParametersManager Tests: persistence and notifications.
- NotificationManager Tests: thread safety and robustness.
- UnifiedTimeModule Tests: parameter-driven calendar regeneration.
- Edge-Case Tests: invalid inputs, corrupted files, large tick advances.

Run all tests with: `pytest` Use markers: `pytest -m integration`, `pytest -m edge`.