Contents

OH Parser - Complete Documentation **1**
Table of Contents . . . 1
Overview . . . 2
Key Features . . . 2
Design Principles . . . 2
Project Structure . . . 2
Installation . . . 2
Quick Start . . . 3
OH Profile JSON Structure . . . 3
Top-Level Structure . . . 3
EMG Nested Structure (3 Levels) . . . 4
API Reference . . . 4
Loading Functions . . . 4
Inspection Functions . . . 4
Extraction Functions . . . 5
Path Resolution Functions . . . 6
Filtering . . . 6
Usage Examples . . . 7
Example 1: Weekly EMG Summary (Wide Format) . . . 7
Example 2: All Session-Level Data (Long Format) . . . 7
Example 3: Filter by Subjects and Date Range . . . 7
Example 4: Check Data Availability . . . 8
Example 5: Manual Path Navigation . . . 8
Path Syntax & Wildcards . . . 8
Dot Notation . . . 8
Wildcards in `value_paths` . . . 8
Level Names (Implicit Wildcards) . . . 9
Exclusion Patterns . . . 9
CLI Interface . . . 9
EMG Data Reference . . . 9
Metric Groups . . . 9
Aggregation Levels . . . 10
Pipeline Architecture . . . 10
Data Flow . . . 11
Notes . . . 11

OH Parser - Complete Documentation

Comprehensive guide for extracting data from Occupational Health (OH) profile JSON files into pandas DataFrames.

Table of Contents

1. Overview
2. Project Structure
3. Installation
4. Quick Start
5. OH Profile JSON Structure
6. API Reference
7. Usage Examples

8. Path Syntax & Wildcards
9. CLI Interface
10. EMG Data Reference
11. Pipeline Architecture

Overview

OH Parser is a Python library for extracting data from Occupational Health (OH) profile JSON files into pandas DataFrames for statistical analysis.

Key Features

- **Load** JSON profiles from a directory (`load_profiles`)
- **Inspect** profile structure with tree visualization (`inspect_profile`)
- **Extract** data using dot-notation paths:
  - `extract()` -> Wide format (one row per subject)
  - `extract_nested()` -> Long format (one row per session/date/side)
  - `extract_flat()` -> Flatten all nested keys
- **Filter** by subjects, date ranges, groups, or data availability (`create_filters`)
- **CLI** for quick exploration without writing code

Design Principles

1. **Pure function-based** - No classes, just functions and dictionaries
2. **Dot-notation paths** - Navigate nested JSON with strings like `"a.b.c"`
3. **Wildcard support** - Extract all keys with `"EMG_intensity.*"`
4. **Minimal dependencies** - Only requires pandas

Project Structure

```
oh_parser/
|-- __init__.py          # Public API exports
|-- __main__.py          # CLI entrypoint (python -m oh_parser)
|-- cli.py               # Command-line interface
|-- loader.py            # Load OH profile JSON files
|-- path_resolver.py     # Dot-notation path navigation
|-- filters.py           # Subject/date filtering
|-- extract.py           # DataFrame extraction functions
+-- utils.py             # Utility functions
```

Installation

Copy the `oh_parser/` folder to your project, then install dependencies:

```
pip install pandas
```

Or with requirements file:

```
pip install -r requirements.txt
```

Dependencies: pandas >= 1.5.0, Python >= 3.9

Quick Start

```python
from oh_parser import load_profiles, list_subjects, inspect_profile, extract, extract_nested

# 1. Load all profiles
profiles = load_profiles("/path/to/OH_profiles/")

# 2. List subjects
subjects = list_subjects(profiles)  # ['80', '81', '82', ...]

# 3. Inspect structure
inspect_profile(profiles[subjects[0]], max_depth=4)

# 4. Extract specific values (wide format - one row per subject)
df = extract(profiles, paths={
    "p50_L": "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p50",
    "p50_R": "sensor_metrics.emg.EMG_weekly_metrics.right.EMG_apdf.active.p50",
})

# 5. Extract nested data (long format - one row per session)
df = extract_nested(
    profiles,
    base_path="sensor_metrics.emg",
    level_names=["date", "session", "side"],
    value_paths=["EMG_intensity.*", "EMG_rest_recovery.*"],
    exclude_patterns=["EMG_daily_metrics", "EMG_weekly_metrics"],
)
```

OH Profile JSON Structure

Each subject has one file: `{subject_id}_OH_profile.json`

Top-Level Structure

```json
{
  "meta_data": { "age": 45, "gender": "M", ... },
  "single_instance_questionnaires": {
    "personal": { ... },
    "biomechanical": { ... },
    "psychosocial": { ... },
    "environmental": { ... }
  },
  "daily_questionnaires": {
    "workload": { "YYYY-MM-DD": { ... } },
    "pain": { "YYYY-MM-DD": { ... } }
  },
  "sensor_metrics": {
    "sensor_timeline": { ... },
    "human_activities": { ... },
    "heart_rate": { ... },
    "posture": { ... },
    "noise": { ... },
    "emg": { ... },
    "wrist_activities": { ... }
  }
}
```

EMG Nested Structure (3 Levels)

```
sensor_metrics.emg/
|-- {date: DD-MM-YYYY}/                 # Level 1: Recording day
|   |-- {session: HH-MM-SS}/            # Level 2: Session start time
|   |   |-- left/                       # Level 3: Side
|   |   |   |-- EMG_session/            # Session metadata
|   |   |   |-- EMG_intensity/          # Intensity metrics
|   |   |   |-- EMG_apdf/               # APDF percentiles
|   |   |   |-- EMG_rest_recovery/      # Rest/recovery metrics
|   |   |   +-- EMG_relative_bins/      # Relative intensity bins
|   |   +-- right/
|   |       +-- ... (same structure)
|   +-- EMG_daily_metrics/              # Aggregated daily
|       |-- left/ { ... }
|       +-- right/ { ... }
+-- EMG_weekly_metrics/                 # Aggregated weekly
    |-- left/ { ... }
    +-- right/ { ... }
```

API Reference

Loading Functions

`load_profiles(directory, subject_ids=None, verbose=True) -> Dict[str, dict]` Load all OH profiles from a directory.

Parameter	Type	Description
`directory`	`str | Path`	Path to directory containing OH profiles
`subject_ids`	`List[str]`	Optional: only load specific subjects
`verbose`	`bool`	Print loading progress

```
profiles = load_profiles("/path/to/OH_profiles/")
profiles = load_profiles("/path/to/OH_profiles/", subject_ids=["103", "104"])
```

`list_subjects(profiles) -> List[str]` Get sorted list of subject IDs (sorted numerically).

`get_profile(profiles, subject_id) -> dict | None` Get a single profile by subject ID. Returns `None` if not found.

`load_profile(filepath) -> dict` Load a single OH profile JSON file.

Inspection Functions

`inspect_profile(profile, base_path="", max_depth=4, show_values=False)` Pretty-print the structure of a profile as a tree.

```
inspect_profile(profile)
inspect_profile(profile, base_path="sensor_metrics.emg", max_depth=3)
```

`get_available_paths(profile, base_path="", max_depth=6) -> List[str]` Get all dot-notation paths available in a profile.

```
paths = get_available_paths(profile)
# ['meta_data.age', 'sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p50', ...]
```

`summarize_profiles(profiles) -> pd.DataFrame` Generate summary showing data availability across subjects.

```
summary = summarize_profiles(profiles)
# Columns: subject_id, has_meta_data, has_emg, has_EMG_weekly_metrics, ...
```

Extraction Functions

`extract(profiles, paths, filters=None, include_subject_id=True) -> pd.DataFrame` Extract specific paths into **wide-format DataFrame** (one row per subject).

Parameter	Type	Description
`profiles`	`Dict[str, dict]`	Loaded profiles
`paths`	`Dict[str, str]`	Mapping: column name -> dot-notation path
`filters`	`dict`	Optional filters from `create_filters()`
`include_subject_id`	`bool`	Include subject_id column

```
df = extract(profiles, paths={
    "age": "meta_data.age",
    "emg_p50": "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p50",
})
# Output: subject_id | age | emg_p50
```

`extract_nested(profiles, base_path, level_names, value_paths=None, filters=None, exclude_patterns=None, flatten_values=True) -> pd.DataFrame` Extract nested structures into **long-format DataFrame** (one row per leaf node).

Parameter	Type	Description
`profiles`	`Dict[str, dict]`	Loaded profiles
`base_path`	`str`	Starting path (e.g., `"sensor_metrics.emg"`)
`level_names`	`List[str]`	Names for nesting levels
`value_paths`	`List[str]`	Paths to extract (supports `*` wildcard)
`exclude_patterns`	`List[str]`	Glob patterns to skip (e.g., `["EMG_*_metrics"]`)
`filters`	`dict`	Optional filters from `create_filters()`
`flatten_values`	`bool`	Flatten nested dicts into columns

```
df = extract_nested(
    profiles,
    base_path="sensor_metrics.emg",
    level_names=["date", "session", "side"],
    value_paths=["EMG_intensity.*", "EMG_rest_recovery.*"],
    exclude_patterns=["EMG_daily_metrics", "EMG_weekly_metrics"],
)
# Output: subject_id | date | session | side | EMG_intensity.mean_percent_mvc | ...
```

`extract_flat(profiles, base_path, filters=None, max_depth=10) -> pd.DataFrame` Flatten everything under a path into wide-format (one row per subject).

```
df = extract_flat(profiles, base_path="sensor_metrics.emg.EMG_weekly_metrics")
# Columns: subject_id, left.EMG_apdf.active.p10, left.EMG_apdf.active.p50, ...
```

Path Resolution Functions

`resolve_path(data, path, default=None)` Get value from nested dict using dot-notation.

```
value = resolve_path(profile, "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p50")
# Returns: 12.5 (or None if path doesn't exist)
```

`path_exists(data, path) -> bool` Check if a path exists.

`list_keys_at_path(data, path) -> List[str]` List all keys at a given path.

```
keys = list_keys_at_path(profile, "sensor_metrics.emg")
# ['23-09-2025', '24-09-2025', 'EMG_weekly_metrics']
```

Filtering

`create_filters(...) -> Dict[str, Any]` Create a filters dictionary for controlling extraction.

Parameter	Type	Description
`subject_ids`	`List[str]`	Include only these subjects
`exclude_subjects`	`List[str]`	Exclude these subjects
`groups`	`List[str]`	Filter by `meta_data.group`
`date_range`	`Tuple[str, str]`	`(start, end)` in YYYY-MM-DD format
`require_keys`	`List[str]`	Only include subjects with these paths
`custom_filter`	`Callable`	Custom `(subject_id, profile) -> bool`

```
from oh_parser import create_filters

filters = create_filters(
    subject_ids=["103", "104", "105"],
    date_range=("2025-09-01", "2025-09-30"),
    require_keys=["sensor_metrics.emg.EMG_weekly_metrics"],
)

df = extract_nested(profiles, ..., filters=filters)
```

Usage Examples

Example 1: Weekly EMG Summary (Wide Format)

```python
from oh_parser import load_profiles, extract

profiles = load_profiles("/path/to/OH_profiles/")

df = extract(profiles, paths={
    # Left side
    "p10_L": "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p10",
    "p50_L": "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p50",
    "p90_L": "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p90",
    "rest_pct_L": "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_rest_recovery.rest_percent",
    # Right side
    "p10_R": "sensor_metrics.emg.EMG_weekly_metrics.right.EMG_apdf.active.p10",
    "p50_R": "sensor_metrics.emg.EMG_weekly_metrics.right.EMG_apdf.active.p50",
    "p90_R": "sensor_metrics.emg.EMG_weekly_metrics.right.EMG_apdf.active.p90",
    "rest_pct_R": "sensor_metrics.emg.EMG_weekly_metrics.right.EMG_rest_recovery.rest_percent",
})
```

Example 2: All Session-Level Data (Long Format)

```python
from oh_parser import load_profiles, extract_nested

profiles = load_profiles("/path/to/OH_profiles/")

df = extract_nested(
    profiles,
    base_path="sensor_metrics.emg",
    level_names=["date", "session", "side"],
    value_paths=[
        "EMG_session.*",
        "EMG_intensity.*",
        "EMG_apdf.active.*",
        "EMG_rest_recovery.*",
    ],
    exclude_patterns=["EMG_daily_metrics", "EMG_weekly_metrics"],
)
# Columns: subject_id, date, session, side, EMG_session.duration_s, ...
```

Example 3: Filter by Subjects and Date Range

```python
from oh_parser import load_profiles, extract_nested, create_filters

profiles = load_profiles("/path/to/OH_profiles/")

filters = create_filters(
    subject_ids=["80", "81", "82"],
    date_range=("2025-09-23", "2025-09-26"),
)

df = extract_nested(
    profiles,
    base_path="sensor_metrics.emg",
    level_names=["date", "session", "side"],
    value_paths=["EMG_intensity.mean_percent_mvc"],
    exclude_patterns=["EMG_daily_metrics", "EMG_weekly_metrics"],
    filters=filters,
)
```

Example 4: Check Data Availability

```
from oh_parser import load_profiles, summarize_profiles, create_filters

profiles = load_profiles("/path/to/OH_profiles/")

# Check which subjects have EMG data
summary = summarize_profiles(profiles)
print(summary[summary['has_emg'] == True])

# Filter to only subjects with weekly EMG
filters = create_filters(
    require_keys=["sensor_metrics.emg.EMG_weekly_metrics.left"],
)
df = extract(profiles, paths={...}, filters=filters)
```

Example 5: Manual Path Navigation

```
from oh_parser import load_profiles, get_profile, resolve_path, list_keys_at_path, path_exists

profiles = load_profiles("/path/to/OH_profiles/")
profile = get_profile(profiles, "103")

# Check if path exists
if path_exists(profile, "sensor_metrics.emg.EMG_weekly_metrics"):
    print("Has weekly EMG!")

# Get a specific value
p50 = resolve_path(profile, "sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p50")

# List available dates
dates = list_keys_at_path(profile, "sensor_metrics.emg")
```

Path Syntax & Wildcards

Dot Notation

Navigate nested structures with dots:

```
sensor_metrics.emg.EMG_weekly_metrics.left.EMG_apdf.active.p50
```

Wildcards in `value_paths`

Use `.*` to extract all keys under a path:

```
value_paths=["EMG_intensity.*"]
# Extracts: EMG_intensity.mean_percent_mvc, EMG_intensity.max_percent_mvc, ...

value_paths=["EMG_apdf.active.*"]
# Extracts: EMG_apdf.active.p10, EMG_apdf.active.p50, EMG_apdf.active.p90
```

Level Names (Implicit Wildcards)

The `level_names` parameter creates wildcards for dynamic keys:

```
base_path="sensor_metrics.emg"
level_names=["date", "session", "side"]
# Iterates: sensor_metrics.emg.{date}.{session}.{side}
```

Exclusion Patterns

Glob-style patterns to skip keys:

```
exclude_patterns=["EMG_*_metrics"]    # Skips EMG_daily_metrics, EMG_weekly_metrics
exclude_patterns=["*_aggregate"]      # Skips anything ending in _aggregate
```

CLI Interface

Quick exploration without writing code:

```
# Basic info (default path)
python -m oh_parser --dir /path/to/OH_profiles

# List all subject IDs
python -m oh_parser --dir /path/to/OH_profiles --list

# Inspect a subject's profile structure
python -m oh_parser --dir /path/to/OH_profiles --inspect 103 --depth 5

# List all available paths
python -m oh_parser --dir /path/to/OH_profiles --paths 103

# Show data availability summary
python -m oh_parser --dir /path/to/OH_profiles --summary

# Quiet mode (suppress loading messages)
python -m oh_parser --dir /path/to/OH_profiles --quiet
```

EMG Data Reference

Metric Groups

`EMG_session` - Session Metadata

Key	Type	Unit	Description
`duration_s`	float	seconds	Total recording duration
`mvc_peak`	float	mV	MVC value used for normalization
`active_duration_s`	float	seconds	Time above rest threshold

`EMG_intensity` - Intensity Metrics

Key	Type	Unit	Description
`mean_percent_mvc`	float	%MVC	Mean amplitude
`max_percent_mvc`	float	%MVC	Peak amplitude
`min_percent_mvc`	float	%MVC	Minimum amplitude
`iemg_percent_seconds`	float	%MVC*s	Integrated EMG (area under curve)

`EMG_apdf` - Amplitude Probability Distribution Function

Key	Type	Unit	Description
`full.p10`	float	%MVC	10th percentile (all samples)
`full.p50`	float	%MVC	50th percentile / median
`full.p90`	float	%MVC	90th percentile
`active.p10`	float	%MVC	10th percentile (active only, >=0.5% MVC)
`active.p50`	float	%MVC	50th percentile (active only)
`active.p90`	float	%MVC	90th percentile (active only)

`EMG_rest_recovery` - Rest/Recovery Metrics

Key	Type	Unit	Description
`rest_percent`	float	%	Time below 0.5% MVC threshold
`gap_frequency_per_minute`	float	gaps/min	Micro-break frequency
`max_sustained_activity_s`	float	seconds	Longest continuous active period
`gap_count`	int	count	Total number of rest gaps

`EMG_relative_bins` - Relative Intensity Distribution

Key	Type	Unit	Description
`below_usual_pct`	float	%	Time below weekly P10
`typical_low_pct`	float	%	Time between P10-P50
`typical_high_pct`	float	%	Time between P50-P90
`high_for_you_pct`	float	%	Time above weekly P90

Aggregation Levels

Level	Path	Description
Session	`emg.{date}.{session}.{side}`	Per-recording metrics
Daily	`emg.{date}.EMG_daily_metrics.{side}`	Duration-weighted daily average
Weekly	`emg.EMG_weekly_metrics.{side}`	Duration-weighted weekly average

Pipeline Architecture

```
+-----------------------------------------------------------------------------+
|                           OH PARSER PIPELINE                                |
|                                                                             |
|   JSON Files     ->    Load     ->    Filter    ->    Navigate    -> DataFrame|
|                                                                             |
|   *_OH_profile.json   loader.py    filters.py   path_resolver.py  extract.py|
+-----------------------------------------------------------------------------+
```

Data Flow

```
/path/to/OH_profiles/
|-- 80_OH_profile.json  --+
|-- 81_OH_profile.json    +--> load_profiles() --> {"80": {...}, "81": {...}}
+-- 82_OH_profile.json  --+                                   |
                                                              v
                                                 apply_subject_filters()
                                                              |
                                                              v
                                                    Filtered Profiles
                                                              |
                                  +---------------------------+---------------------------+
                                  v                           v                           v
                             extract()              extract_nested()          extract_flat()
                                  |                           |                           |
                                  v                           v                           v
                              Wide DF                    Long DF              Flat Wide DF
                           (1 row/subject)          (1 row/session)       (all keys as cols)
```

Notes

- **Date format**: EMG paths use `DD-MM-YYYY` format
- **Time format**: Session times use `HH-MM-SS` format
- **Null values**: Some metrics may be `None` (e.g., relative bins at weekly level)
- **Sides**: EMG data has separate entries for `left` and `right`
- **macOS**: Hidden `._*` files are automatically skipped when loading

---

Generated for oh_parser module - PrevOccupAI+ Project