`processes.py` Documentation

Process Management and JSON Persistence Module

February 13, 2026

Warning: This documentation was generated by an AI assistant. However I have reviewed this file and made changes accordingly if needed.

Abstract

`processes.py` is a Python module designed for process management with JSON persistence capabilities. It provides three core classes: `Mindmap` for process relationship mapping, `TimeStamp` for temporal tracking with disruptions, and `Process` for comprehensive process management with automatic instance tracking and O(1) lookup by unique identifier.

Contents

1 Overview

The processes.py module implements a process management system with the following key features:

- **Automatic Instance Tracking**: Uses weakref.WeakSet to track all created Process instances without manual registration
- **O(1) UID Lookup**: Dictionary-based JSON storage enables constant-time process retrieval
- **Temporal Disruption Handling**: TimeStamp class supports nested disruptions (pauses, async interruptions)
- **Merge-based Storage**: storeAll() preserves existing data while updating changed processes
- **Graph-based Relationships**: Mindmap class maps process dependencies and connections

2 Dependencies

The module requires only Python standard library modules:

```
import random    # For UID generation
import json      # For persistence
import weakref   # For automatic instance tracking
import os        # For file operations
```

3 Class Reference

3.1 Mindmap

The Mindmap class implements a directed graph structure for mapping process relationships.

3.1.1 Constructor

```
class Mindmap:
    def __init__(self, map: dict):
        """
        A mindmap can map the schematics of a process.
        The map dictionary includes nodes (process names)
        and edges (connections between nodes).
        """
        self.map = map
```

3.1.2 Methods

3.1.3 Connection Direction

The connect() method supports three connection types via the direction parameter:

- direction=None: Bidirectional connection (both nodes reference each other)
- direction=False: Source to node (srcNode → node)
- direction=True: Node to source (node → srcNode)

Method	Returns	Description
`nodes()`	`dict_keys`	Returns all node names in the mindmap
`edges()`	`dict_values`	Returns all edge connections
`connections(node)`	`list`	Returns nodes connected to given source node
`connected(src, node)`	`bool`	Checks if two nodes are connected
`connect(src, node, direction)`	`None`	Creates connection with optional directionality

3.2 TimeStamp

The `TimeStamp` class handles temporal tracking with support for nested disruptions.

3.2.1 Constructor

```
class TimeStamp:
    def __init__(self, _init: int, _term: int, _dis=None):
        """
        _init: time of initiation (Unix timestamp)
        _term: time of termination (Unix timestamp, -1 = ongoing)
        _dis: disruptions dict {"type": TimeStamp}
        """
        self.ts = {"initialized": _init, "terminated": _term}
        self.ts["disruptions"] = []

        if _dis is not None:
            for i in _dis:
                self.ts["disruptions"].append({i: _dis[i]})
```

3.2.2 Instance Methods

Method	Returns	Description
`add_disruption(type, ts)`	`None`	Adds a disruption TimeStamp to this timestamp
`serialize_disruptions()`	`list`	Converts disruptions to JSON-serializable format
`to_hirearchial_dict()`	`dict`	Returns complete timeline dictionary

3.2.3 Static Methods

```
@staticmethod
def de_serialize(dis: list) -> dict:
    """
    Reconstructs TimeStamp objects from JSON disruption list.
    Returns dict of {disruption_type: TimeStamp}.
    """
```

3.3 Process

The `Process` class extends `TimeStamp` with process management capabilities and automatic instance tracking.

3.3.1 Class-Level Instance Tracking

```
class Process(TimeStamp):
    __instances = weakref.WeakSet()
    # Automatically tracks all Process instances
```

The `weakref.WeakSet` ensures:

- Automatic registration in `__init__`
- No memory leaks (weak references allow garbage collection)
- No manual tracking required

3.3.2 Constructor

```
def __init__(self, name: str, layer: int, _init: int, _term: int, _dis=
    None):
    """
    name: Process name/identifier
    layer: Hierarchical layer (0 = parallel processing allowed)
    _init: Start time (Unix timestamp)
    _term: End time (-1 = ongoing)
    _dis: Disruptions dictionary

    Auto-generates UID: "{layer}-{4-digit-hex}"
    Auto-registers to __instances
    """
```

3.3.3 Instance Methods

Method	Returns	Description
`to_hirearchial_dict()`	`dict`	Full process dictionary with timeline
`to_storage_dict()`	`tuple`	(uid, dict) for dict storage
`write(ofile)`	`str`	Save/update single process to JSON

3.3.4 Class Methods

Method	Returns	Description
`load_dict(data)`	`Process`	Create Process from dictionary
`storeAll(ofile)`	`str`	Merge all instances to JSON file
`loadAll(ofile)`	`list`	Load all processes from file
`load(uid, ofile)`	`Process|None`	O(1) lookup by UID

3.3.5 Static Methods

```python
@staticmethod
def get_layer_from_uid(uid: str) -> int:
    """Extracts layer number from UID format 'layer-hex'"""
    # Example: "1-a3f2" -> 1

@staticmethod
def remove(uid: str, ofile: str) -> bool:
    """Remove process by UID from storage"""

@staticmethod
def clear_storage(ofile: str) -> None:
    """Clear all processes from storage"""
```

4 JSON Storage Format

The module uses a dictionary-based JSON structure for O(1) UID lookup:

```
{
  "processes": {
    "1-a3f2": {
      "name": "Training",
      "uid": "1-a3f2",
      "timeline": {
        "initialized": 1000,
        "terminated": 5000,
        "disruptions": [
          {
            "pause": {
              "initialized": 2000,
              "terminated": 2500
            }
          }
        ]
      }
    },
    "2-b7c9": {
      "name": "Validation",
      "uid": "2-b7c9",
      "timeline": {
        "initialized": 6000,
        "terminated": -1,
        "disruptions": []
      }
    }
  }
}
```

Key Design Decision: Using UID as dictionary key enables $O(1)$ lookup vs $O(n)$ list iteration.

5 Usage Examples

5.1 Creating and Saving Processes

```python
from processes import Process, TimeStamp

# Create processes (auto-tracked)
p1 = Process("Training", layer=1, _init=0, _term=100)
p2 = Process("Validation", layer=2, _init=100, _term=200)

# Add disruption
pause = TimeStamp(50, 75)
p1.add_disruption("pause", pause)

# Save single process
p1.write("processes.json")

# Save all tracked instances
Process.storeAll("processes.json")
```

5.2 Loading Processes

```python
# O(1) lookup by UID
p = Process.load("1-a3f2", "processes.json")
if p:
    print(f"Found: {p.name}")

# Load all processes
all_processes = Process.loadAll("processes.json")

# Load from dictionary (for custom processing)
data = {"name": "Test", "uid": "1-1234",
        "timeline": {"initialized": 0, "terminated": 100}}
p = Process.load_dict(data)
```

5.3 Removing Processes

```python
# Remove single process
success = Process.remove("1-a3f2", "processes.json")

# Clear all storage
Process.clear_storage("processes.json")
```

5.4 Using Mindmap for Dependencies

```python
from processes import Mindmap

# Create process dependency map
mindmap = Mindmap({
    "DataLoading": [],
    "Preprocessing": ["DataLoading"],
    "Training": ["Preprocessing"],
    "Validation": ["Training"]
})

# Check dependencies
print(mindmap.connections("Training"))  # ["Preprocessing"]
print(mindmap.connected("DataLoading", "Training"))  # False
```

6 Method Reference by Category

6.1 Serialization

- `TimeStamp.serialize_disruptions()` → JSON-serializable disruption list
- `TimeStamp.to_hirearchial_dict()` → Complete timeline dict
- `Process.to_hirearchial_dict()` → Complete process dict
- `Process.to_storage_dict()` → (uid, dict) tuple

6.2 Deserialization

- `TimeStamp.de_serialize(list)` → `{type: TimeStamp}` dict
- `Process.load_dict(dict)` → `Process` instance

6.3 Persistence

- `Process.write(ofile)` → Save/update single process
- `Process.storeAll(ofile)` → Merge all instances to file
- `Process.loadAll(ofile)` → Load all as list
- `Process.load(uid, ofile)` → O(1) lookup
- `Process.remove(uid, ofile)` → Remove by UID
- `Process.clear_storage(ofile)` → Clear all

7 Notes and Limitations

- **Thread Safety:** File operations are not atomic; concurrent writes may cause data loss
- **Memory Management:** `WeakSet` auto-removes unreferenced instances from tracking
- **UID Collisions:** 4-digit hex provides 65536 combinations; collision probability low but non-zero
- **JSON Limitations:** All timestamps stored as integers (Unix epoch)
- **Disruption Nesting:** Disruptions can contain disruptions recursively