```
import toml
import yaml
import asyncio
import concurrent.futures
import json
import os
from concurrent.futures import ThreadPoolExecutor
from datetime import datetime
from typing import Any, Dict, List, Optional, Callable



import psutil


try:
    import gzip
except ImportError as error:
    print(f"Error importing gzip: {error}")
# from pydantic import BaseModel



class BaseStructure:
    """Base structure.



    Attributes:
        name (Optional[str]): _description_
```

description (Optional[str]): _description_

save_metadata (bool): _description_

save_artifact_path (Optional[str]): _description_

save_metadata_path (Optional[str]): _description_

save_error_path (Optional[str]): _description_

Methods:

run: _description_

save_to_file: _description_

load_from_file: _description_

save_metadata: _description_

load_metadata: _description_

log_error: _description_

save_artifact: _description_

load_artifact: _description_

log_event: _description_

run_async: _description_

save_metadata_async: _description_

load_metadata_async: _description_

log_error_async: _description_

save_artifact_async: _description_

load_artifact_async: _description_

log_event_async: _description_

asave_to_file: _description_

aload_from_file: _description_

run_in_thread: _description_

```
        save_metadata_in_thread: _description_
        run_concurrent: _description_
        compress_data: _description_
        decompres_data: _description_
        run_batched: _description_
        load_config: _description_
        backup_data: _description_
        monitor_resources: _description_
        run_with_resources: _description_
        run_with_resources_batched: _description_

    Examples:
    >>> base_structure = BaseStructure()
    >>> base_structure
                        BaseStructure(name=None, description=None, save_metadata=True,
save_artifact_path='./artifacts', save_metadata_path='./metadata', save_error_path='./errors')
    """

    def __init__(
        self,
        name: Optional[str] = None,
        description: Optional[str] = None,
        save_metadata_on: bool = True,
        save_artifact_path: Optional[str] = "./artifacts",
        save_metadata_path: Optional[str] = "./metadata",
        save_error_path: Optional[str] = "./errors",
```

```
    workspace_dir: Optional[str] = "./workspace",
):
    super().__init__()
    self.name = name
    self.description = description
    self.save_metadata_on = save_metadata_on
    self.save_artifact_path = save_artifact_path
    self.save_metadata_path = save_metadata_path
    self.save_error_path = save_error_path
    self.workspace_dir = workspace_dir

def run(self, *args, **kwargs):
    """Run the structure."""

def save_to_file(self, data: Any, file_path: str):
    """Save data to file.

    Args:
        data (Any): _description_
        file_path (str): _description_
    """
    with open(file_path, "w") as file:
        json.dump(data, file)

def load_from_file(self, file_path: str) -> Any:
    """Load data from file.
```

```
    Args:
        file_path (str): _description_

    Returns:
        Any: _description_
    """
    with open(file_path) as file:
        return json.load(file)

def save_metadata(self, metadata: Dict[str, Any]):
    """Save metadata to file.

    Args:
        metadata (Dict[str, Any]): _description_
    """
    if self.save_metadata:
        file_path = os.path.join(
            self.save_metadata_path, f"{self.name}_metadata.json"
        )
        self.save_to_file(metadata, file_path)

def load_metadata(self) -> Dict[str, Any]:
    """Load metadata from file.

    Returns:
```

```
        Dict[str, Any]: _description_
    """
    file_path = os.path.join(
        self.save_metadata_path, f"{self.name}_metadata.json"
    )
    return self.load_from_file(file_path)

def log_error(self, error_message: str):
    """Log error to file.

    Args:
        error_message (str): _description_
    """
    file_path = os.path.join(
        self.save_error_path, f"{self.name}_errors.log"
    )
    with open(file_path, "a") as file:
        file.write(f"{error_message}\n")

def save_artifact(self, artifact: Any, artifact_name: str):
    """Save artifact to file.

    Args:
        artifact (Any): _description_
        artifact_name (str): _description_
    """
```

```
    file_path = os.path.join(
        self.save_artifact_path, f"{artifact_name}.json"
    )
    self.save_to_file(artifact, file_path)

def load_artifact(self, artifact_name: str) -> Any:
    """Load artifact from file.

    Args:
        artifact_name (str): _description_

    Returns:
        Any: _description_
    """
    file_path = os.path.join(
        self.save_artifact_path, f"{artifact_name}.json"
    )
    return self.load_from_file(file_path)

def _current_timestamp(self):
    """Current timestamp.

    Returns:
        _type_: _description_
    """
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
def log_event(
    self,
    event: str,
    event_type: str = "INFO",
):
    """Log event to file.

    Args:
        event (str): _description_
        event_type (str, optional): _description_. Defaults to "INFO".
    """
    timestamp = self._current_timestamp()
    log_message = f"[{timestamp}] [{event_type}] {event}\n"
    file = os.path.join(
        self.save_metadata_path, f"{self.name}_events.log"
    )
    with open(file, "a") as file:
        file.write(log_message)

async def run_async(self, *args, **kwargs):
    """Run the structure asynchronously."""
    loop = asyncio.get_event_loop()
    return await loop.run_in_executor(
        None, self.run, *args, **kwargs
    )
```

```
async def save_metadata_async(self, metadata: Dict[str, Any]):
    """Save metadata to file asynchronously.

    Args:
        metadata (Dict[str, Any]): _description_
    """
    loop = asyncio.get_event_loop()
    return await loop.run_in_executor(
        None, self.save_metadata, metadata
    )

async def load_metadata_async(self) -> Dict[str, Any]:
    """Load metadata from file asynchronously.

    Returns:
        Dict[str, Any]: _description_
    """
    loop = asyncio.get_event_loop()
    return await loop.run_in_executor(None, self.load_metadata)

async def log_error_async(self, error_message: str):
    """Log error to file asynchronously.

    Args:
        error_message (str): _description_
```

```
    """
    loop = asyncio.get_event_loop()
    return await loop.run_in_executor(
        None, self.log_error, error_message
    )

async def save_artifact_async(
    self, artifact: Any, artifact_name: str
):
    """Save artifact to file asynchronously.

    Args:
        artifact (Any): _description_
        artifact_name (str): _description_
    """
    loop = asyncio.get_event_loop()
    return await loop.run_in_executor(
        None, self.save_artifact, artifact, artifact_name
    )

async def load_artifact_async(self, artifact_name: str) -> Any:
    """Load artifact from file asynchronously.

    Args:
        artifact_name (str): _description_
```

```
    Returns:

        Any: _description_

    """

    loop = asyncio.get_event_loop()

    return await loop.run_in_executor(

        None, self.load_artifact, artifact_name

    )


async def log_event_async(

    self,

    event: str,

    event_type: str = "INFO",

):

    """Log event to file asynchronously.


    Args:

        event (str): _description_

        event_type (str, optional): _description_. Defaults to "INFO".

    """

    loop = asyncio.get_event_loop()

    return await loop.run_in_executor(

        None, self.log_event, event, event_type

    )


async def asave_to_file(

    self, data: Any, file: str, *args, **kwargs
```

```
):
    """Save data to file asynchronously.

    Args:
        data (Any): _description_
        file (str): _description_
    """
    await asyncio.to_thread(
        self.save_to_file,
        data,
        file,
        *args,
    )

async def aload_from_file(
    self,
    file: str,
) -> Any:
    """Async load data from file.

    Args:
        file (str): _description_

    Returns:
        Any: _description_
    """
```

```
    return await asyncio.to_thread(self.load_from_file, file)


def run_in_thread(self, *args, **kwargs):
    """Run the structure in a thread."""
    with concurrent.futures.ThreadPoolExecutor() as executor:
        return executor.submit(self.run, *args, **kwargs)


def save_metadata_in_thread(self, metadata: Dict[str, Any]):
    """Save metadata to file in a thread.

    Args:
        metadata (Dict[str, Any]): _description_
    """
    with concurrent.futures.ThreadPoolExecutor() as executor:
        return executor.submit(self.save_metadata, metadata)


def run_concurrent(self, *args, **kwargs):
    """Run the structure concurrently."""
    return asyncio.run(self.run_async(*args, **kwargs))


def compress_data(
    self,
    data: Any,
) -> bytes:
    """Compress data.
```

```
    Args:
        data (Any): _description_

    Returns:
        bytes: _description_
    """
    return gzip.compress(json.dumps(data).encode())

def decompres_data(self, data: bytes) -> Any:
    """Decompress data.

    Args:
        data (bytes): _description_

    Returns:
        Any: _description_
    """
    return json.loads(gzip.decompress(data).decode())

def run_batched(
    self,
    batched_data: List[Any],
    batch_size: int = 10,
    *args,
    **kwargs,
):
```

```
    """Run batched data.

    Args:
        batched_data (List[Any]): _description_
        batch_size (int, optional): _description_. Defaults to 10.

    Returns:
        _type_: _description_
    """
    with ThreadPoolExecutor(max_workers=batch_size) as executor:
        futures = [
            executor.submit(self.run, data)
            for data in batched_data
        ]
        return [future.result() for future in futures]

def load_config(
    self, config: str = None, *args, **kwargs
) -> Dict[str, Any]:
    """Load config from file.

    Args:
        config (str, optional): _description_. Defaults to None.

    Returns:
        Dict[str, Any]: _description_
```

```
    """
    return self.load_from_file(config)


def backup_data(
    self, data: Any, backup_path: str = None, *args, **kwargs
):
    """Backup data to file.

    Args:
        data (Any): _description_
        backup_path (str, optional): _description_. Defaults to None.
    """
    timestamp = self._current_timestamp()
    backup_file_path = f"{backup_path}/{timestamp}.json"
    self.save_to_file(data, backup_file_path)


def monitor_resources(self):
    """Monitor resource usage."""
    memory = psutil.virtual_memory().percent
    cpu_usage = psutil.cpu_percent(interval=1)
    self.log_event(
        f"Resource usage - Memory: {memory}%, CPU: {cpu_usage}%"
    )


def run_with_resources(self, *args, **kwargs):
    """Run the structure with resource monitoring."""
```

```
    self.monitor_resources()
    return self.run(*args, **kwargs)

def run_with_resources_batched(
    self,
    batched_data: List[Any],
    batch_size: int = 10,
    *args,
    **kwargs,
):
    """Run batched data with resource monitoring.

    Args:
        batched_data (List[Any]): _description_
        batch_size (int, optional): _description_. Defaults to 10.

    Returns:
        _type_: _description_
    """
    self.monitor_resources()
    return self.run_batched(
        batched_data, batch_size, *args, **kwargs
    )

def _serialize_callable(
    self, attr_value: Callable
```

```
) -> Dict[str, Any]:
    """
    Serializes callable attributes by extracting their name and docstring.

    Args:
        attr_value (Callable): The callable to serialize.

    Returns:
        Dict[str, Any]: Dictionary with name and docstring of the callable.
    """
    return {
        "name": getattr(
            attr_value, "__name__", type(attr_value).__name__
        ),
        "doc": getattr(attr_value, "__doc__", None),
    }

def _serialize_attr(self, attr_name: str, attr_value: Any) -> Any:
    """
    Serializes an individual attribute, handling non-serializable objects.

    Args:
        attr_name (str): The name of the attribute.
        attr_value (Any): The value of the attribute.

    Returns:
```

```
        Any: The serialized value of the attribute.
    """
    try:
        if callable(attr_value):
            return self._serialize_callable(attr_value)
        elif hasattr(attr_value, "to_dict"):
            return (
                attr_value.to_dict()
            )  # Recursive serialization for nested objects
        else:
            json.dumps(
                attr_value
            )  # Attempt to serialize to catch non-serializable objects
            return attr_value
    except (TypeError, ValueError):
        return f"<Non-serializable: {type(attr_value).__name__}>"

def to_dict(self) -> Dict[str, Any]:
    """
    Converts all attributes of the class, including callables, into a dictionary.
    Handles non-serializable attributes by converting them or skipping them.

    Returns:
        Dict[str, Any]: A dictionary representation of the class attributes.
    """
    return {
```

```
        attr_name: self._serialize_attr(attr_name, attr_value)
        for attr_name, attr_value in self.__dict__.items()
    }

def to_json(self, indent: int = 4, *args, **kwargs):
    return json.dumps(
        self.to_dict(), indent=indent, *args, **kwargs
    )

def to_yaml(self, indent: int = 4, *args, **kwargs):
    return yaml.dump(
        self.to_dict(), indent=indent, *args, **kwargs
    )

def to_toml(self, *args, **kwargs):
    return toml.dumps(self.to_dict(), *args, **kwargs)

# def model_dump_json(self):
#     logger.info(
#         f"Saving {self.agent_name} model to JSON in the {self.workspace_dir} directory"
#     )

#     create_file_in_folder(
#         self.workspace_dir,
#         f"{self.agent_name}.json",
#         str(self.to_json()),
```

```
#     )

#     return (
#         f"Model saved to {self.workspace_dir}/{self.agent_name}.json"
#     )

# def model_dump_yaml(self):
#     logger.info(
#         f"Saving {self.agent_name} model to YAML in the {self.workspace_dir} directory"
#     )

#     create_file_in_folder(
#         self.workspace_dir,
#         f"{self.agent_name}.yaml",
#         self.to_yaml(),
#     )

#     return (
#         f"Model saved to {self.workspace_dir}/{self.agent_name}.yaml"
#     )
```