```
import json
import os
import subprocess
import time
from datetime import datetime
from typing import Any, Dict, List, Union

from pydantic import BaseModel, Field
from pydantic.v1 import validator

from swarms.utils.file_processing import create_file_in_folder
from swarms.utils.loguru_logger import initialize_logger

logger = initialize_logger(log_folder="main_artifact")


class FileVersion(BaseModel):
    """
    Represents a version of the file with its content and timestamp.
    """

    version_number: int = Field(
        ..., description="The version number of the file"
    )
    content: str = Field(
        ..., description="The content of the file version"
    )
    timestamp: str = Field(
        time.strftime("%Y-%m-%d %H:%M:%S"),
        description="The timestamp of the file version",
    )

    def __str__(self) -> str:
        return f"Version {self.version_number} (Timestamp: {self.timestamp}):\n{self.content}"



class Artifact(BaseModel):
    """
    Represents a file artifact.

    Attributes:
        folder_path
        file_path (str): The path to the file.
        file_type (str): The type of the file.
        contents (str): The contents of the file.
        versions (List[FileVersion]): The list of file versions.
        edit_count (int): The number of times the file has been edited.
    """

    folder_path: str = Field(
        default=os.getenv("WORKSPACE_DIR"),
        description="The path to the folder",
    )
    file_path: str = Field(..., description="The path to the file")
    file_type: str = Field(
        ...,
        description="The type of the file",
        # example=".txt",
    )
    contents: str = Field(
        ..., description="The contents of the file in string format"
    )
    versions: List[FileVersion] = Field(default_factory=list)
    edit_count: int = Field(
        ...,
        description="The number of times the file has been edited",
    )


    @validator("file_type", pre=True, always=True)
    def validate_file_type(cls, v, values):
        if not v:
            file_path = values.get("file_path")
            _, ext = os.path.splitext(file_path)
            if ext.lower() not in [
                ".py",
                ".csv",
                ".tsv",
    ".txt",
    ".json",
    ".xml",
    ".html",
    ".yaml",
    ".yml",
    ".md",
    ".rst",
    ".log",
    ".sh",
    ".bat",
    ".ps1",
    ".psm1",
    ".psd1",
    ".ps1xml",
    ".pssc",
    ".reg",
    ".mof",
    ".mfl",
    ".xaml",
    ".xml",
    ".wsf",
    ".config",
    ".ini",
    ".inf",
    ".json5",
            ".hcl",
            ".tf",
            ".tfvars",
            ".tsv",
            ".properties",
        ]:
            raise ValueError("Unsupported file type")
        return ext.lower()
    return v

def create(self, initial_content: str) -> None:
    """
    Creates a new file artifact with the initial content.
    """
    try:
        self.contents = initial_content
        self.versions.append(
            FileVersion(
                version_number=1,
                content=initial_content,
                timestamp=time.strftime("%Y-%m-%d %H:%M:%S"),
            )
        )
        self.edit_count = 0
    except Exception as e:
        logger.error(f"Error creating artifact: {e}")
```

```
            raise e

    def edit(self, new_content: str) -> None:
        """
        Edits the artifact's content, tracking the change in the version history.
        """
        try:
            self.contents = new_content
            self.edit_count += 1
            new_version = FileVersion(
                version_number=len(self.versions) + 1,
                content=new_content,
                timestamp=time.strftime("%Y-%m-%d %H:%M:%S"),
            )
            self.versions.append(new_version)
        except Exception as e:
            logger.error(f"Error editing artifact: {e}")
            raise e

    def save(self) -> None:
        """
        Saves the current artifact's contents to the specified file path.
        """
        with open(self.file_path, "w") as f:
            f.write(self.contents)
```

```
def load(self) -> None:

    """

    Loads the file contents from the specified file path into the artifact.

    """

    with open(self.file_path, "r") as f:

        self.contents = f.read()

    self.create(self.contents)


def get_version(

    self, version_number: int

) -> Union[FileVersion, None]:

    """

    Retrieves a specific version of the artifact by its version number.

    """

    for version in self.versions:

        if version.version_number == version_number:

            return version

    return None


def get_contents(self) -> str:

    """

    Returns the current contents of the artifact as a string.

    """

    return self.contents


def get_version_history(self) -> str:
    """
    Returns the version history of the artifact as a formatted string.
    """
    return "\n\n".join(
        [str(version) for version in self.versions]
    )

def export_to_json(self, file_path: str) -> None:
    """
    Exports the artifact to a JSON file.

    Args:
        file_path (str): The path to the JSON file where the artifact will be saved.
    """
    with open(file_path, "w") as json_file:
        json.dump(self.dict(), json_file, default=str, indent=4)

@classmethod
def import_from_json(cls, file_path: str) -> "Artifact":
    """
    Imports an artifact from a JSON file.

    Args:
        file_path (str): The path to the JSON file to import the artifact from.

    Returns:
```

```
        Artifact: The imported artifact instance.
    """
    with open(file_path, "r") as json_file:
        data = json.load(json_file)
    # Convert timestamp strings back to datetime objects
    for version in data["versions"]:
        version["timestamp"] = datetime.fromisoformat(
            version["timestamp"]
        )
    return cls(**data)

def get_metrics(self) -> str:
    """
    Returns all metrics of the artifact as a formatted string.

    Returns:
        str: A string containing all metrics of the artifact.
    """
    metrics = (
        f"File Path: {self.file_path}\n"
        f"File Type: {self.file_type}\n"
        f"Current Contents:\n{self.contents}\n\n"
        f"Edit Count: {self.edit_count}\n"
        f"Version History:\n{self.get_version_history()}"
    )
    return metrics
```

```
def to_dict(self) -> Dict[str, Any]:
    """
    Converts the artifact instance to a dictionary representation.
    """
    return self.dict()

@classmethod
def from_dict(cls, data: Dict[str, Any]) -> "Artifact":
    """
    Creates an artifact instance from a dictionary representation.
    """
    try:
        # Convert timestamp strings back to datetime objects if necessary
        for version in data.get("versions", []):
            if isinstance(version["timestamp"], str):
                version["timestamp"] = datetime.fromisoformat(
                    version["timestamp"]
                )
        return cls(**data)
    except Exception as e:
        logger.error(f"Error creating artifact from dict: {e}")
        raise e

def save_as(self, output_format: str) -> None:
    """
```

```
Saves the artifact's contents in the specified format.

Args:
    output_format (str): The desired output format ('.md', '.txt', '.pdf', '.py')

Raises:
    ValueError: If the output format is not supported
"""
supported_formats = {".md", ".txt", ".pdf", ".py"}

if output_format not in supported_formats:
    raise ValueError(
        f"Unsupported output format. Supported formats are: {supported_formats}"
    )

output_path = (
    os.path.splitext(self.file_path)[0] + output_format
)

if output_format == ".pdf":
    self._save_as_pdf(output_path)
else:
    if output_format == ".md":
        # Create the file in the specified folder
        create_file_in_folder(
            self.folder_path,
            self.file_path,
            f"{os.path.basename(self.file_path)}\n\n{self.contents}",
        )

    elif output_format == ".py":
        # Add Python file header
        create_file_in_folder(
            self.folder_path,
            self.file_path,
            f"#{os.path.basename(self.file_path)}\n\n{self.contents}",
        )
    else:  # .txt
        create_file_in_folder(
            self.folder_path,
            self.file_path,
            self.contents,
        )

def _save_as_pdf(self, output_path: str) -> None:
    """
    Helper method to save content as PDF using reportlab
    """
    try:
        from reportlab.lib.pagesizes import letter
        from reportlab.pdfgen import canvas
    except ImportError as e:
            logger.error(f"Error importing reportlab: {e}")

            subprocess.run(["pip", "install", "reportlab"])

            from reportlab.lib.pagesizes import letter

            from reportlab.pdfgen import canvas


        c = canvas.Canvas(output_path, pagesize=letter)

        # Split content into lines

        y = 750  # Starting y position

        for line in self.contents.split("\n"):

            c.drawString(50, y, line)

            y -= 15  # Move down for next line

            if y < 50:  # New page if bottom reached

                c.showPage()

                y = 750

        c.save()



# # Example usage

# artifact = Artifact(file_path="example.txt", file_type=".txt")

# artifact.create("Initial content")

# artifact.edit("First edit")

# artifact.edit("Second edit")

# artifact.save()


# # Export to JSON

# artifact.export_to_json("artifact.json")
# # Import from JSON

# imported_artifact = Artifact.import_from_json("artifact.json")


# # # Get metrics

# print(artifact.get_metrics())




# Testing saving in different artifact types

# Create an artifact

# artifact = Artifact(file_path="/path/to/file", file_type=".txt",contents="",  edit_count=0  )

# artifact.create("This is some content\nWith multiple lines")


# Save in different formats

# artifact.save_as(".md")    # Creates example.md

# artifact.save_as(".txt")   # Creates example.txt

# artifact.save_as(".pdf")   # Creates example.pdf

# artifact.save_as(".py")    # Creates example.py
```