writer_examples

December 12, 2024

[ ]:
```
from readii.io.writers.base_writer import BaseWriter
from readii.utils import logger
import SimpleITK as sitk
from pathlib import Path
import json
from typing import Any
```

[ ]:
```
# Example subclass for writing text files
# Define a concrete subclass of BaseWriter that will handle the saving of a␣
 ↪specific file type
# this is a simple example with no validation or error handling
class TextWriter(BaseWriter):
  def save(self, content: str, **kwargs: Any) -> Path:
    output_path = self.resolve_path(**kwargs)
    with open(output_path, 'w') as f:
      f.write(content)
    return output_path
```

[ ]:
```
example_file_formats = [
  # a placeholder can be of the format {key} or %key
  # {key} is useful for python code, whereas %key is useful for use in CLI or␣
 ↪bash scripts where using {} would be problematic
  "notes_%SubjectID.txt",
  "notes_{SubjectID}.txt",

  # You define the placeholder that you will later pass in as a keyword␣
 ↪argument in the save method
  # By default, the writer automatically generates data for the current "date",␣
 ↪"time", and "date_time"
  # so those can be used as placeholders
  # Every other placeholder needs to be passed in as a keyword argument in the␣
 ↪save method
  "important-file-name_{SubjectID}_{date}.txt",
  "subjects/{SubjectID}/{time}_result.txt",
  "subjects/{SubjectID}_Birthday-{SubjectBirthDate}/data_{date_time}.txt",
]
```

```
# Create text writers with different filename patterns
text_writers = [
  TextWriter(
    root_directory="TRASH/writer_examples/text_data",
    filename_format=fmt
  ) for fmt in example_file_formats
]

# Define some example data to pass to the writers
# this could be extracted from some data source and used to generate the file␣
 ↪names
SubjectID="SUBJ001"
SubjectBirthDate="2022-01-01"

# Test text writers
for writer in text_writers:
  path = writer.save(
    content = "Sample text content", # this is the data that will be written to␣
 ↪the file

    # They key-value pairs can be passed in as keyword arguments, and matched␣
 ↪to placeholders in the filename format
    SubjectID=SubjectID,
    SubjectBirthDate=SubjectBirthDate,

    # If you pass in a key that is not in the filename format, it will be␣
 ↪ignored
    # this can also be seen as `SubjectBirthDate` is only used in one of the␣
 ↪above filename formats
    RandomKey="This will be ignored",
    RandomKey2="This will also be ignored"
  )
  print(f"{writer.__class__.__name__} with format [magenta]'{writer.
 ↪pattern_resolver.formatted_pattern}':")
  print(f"File written to: [green]{path}\n")
```

1 More detailed example

[ ]:
```
import subprocess
import pandas as pd

# Any subclass has to be initialized with a root directory and a filename format
# which might not be obvious at first

class CSVWriter(BaseWriter): # noqa
```

```
  # The save method is the only method that needs to be implemented for the␣
 ↪subclasses of BaseWriter
  def save(self, data: list, **kwargs: Any) -> Path:  # noqa
    output_path = self.resolve_path(**kwargs)
    with output_path.open('w') as f: # noqa
      pd.DataFrame(data).to_csv(f, index=False)
    return output_path

# Make some fake data
subject_data_examples = [
  {
    "PatientID": f"PAT{i:03d}",
    "Modality": f"{MODALITY}",
    "Study": f"Study{j:03d}",
    "DataType": f"{DATA_TYPE}",
  }
  for i in range(1, 4)
  for j in range(1, 3)
  for MODALITY in ["CT", "RTSTRUCT"]
  for DATA_TYPE in ["raw", "processed", "segmented", "labeled"]
]
ROOT_DIRECTORY = Path("TRASH/writer_examples/csv_examples/patient_data")
with CSVWriter(
  root_directory=ROOT_DIRECTORY,
  filename_format="PatientID-{PatientID}/Study-{Study}/{Modality}/
 ↪{DataType}-data.csv"
) as csv_writer:
  # Test CSV writers
  for patient in subject_data_examples:
    path = csv_writer.save(
      data = pd.DataFrame(patient, index=[0]), # just assume that this␣
 ↪dataframe is some real data
      PatientID=patient["PatientID"],
      Study=patient["Study"],
      Modality=patient["Modality"],
      DataType=patient["DataType"]
    )

# run the tree command and capture the output
output = subprocess.check_output(["tree", "-nF",  ROOT_DIRECTORY])
# print(output.decode("utf-8"))
```

Output would look like:

```
TRASH/writer_examples/csv_examples/patient_data/
  PatientID-PAT001/
      Study-Study001/
          CT/
```

```
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
        RTSTRUCT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
    Study-Study002/
        CT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
        RTSTRUCT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
PatientID-PAT002/
    Study-Study001/
        CT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
        RTSTRUCT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
    Study-Study002/
        CT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
        RTSTRUCT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
PatientID-PAT003/
    Study-Study001/
        CT/
            labeled-data.csv
            processed-data.csv
```

```
            raw-data.csv
            segmented-data.csv
        RTSTRUCT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
    Study-Study002/
        CT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv
        RTSTRUCT/
            labeled-data.csv
            processed-data.csv
            raw-data.csv
            segmented-data.csv

22 directories, 48 files
```