Contents

1 CouchDB Integration for Taxon Extraction - Summary

1.1 What Was Created

This integration provides a complete solution for extracting Taxon objects from annotated files stored in CouchDB, using distributed PySpark processing while preserving database metadata throughout the pipeline.

1.2 New Files Created

1.2.1 1. couchdb_file.py

Core module providing CouchDB-aware file reading

Key components: - `CouchDBFile` class: File-like object for CouchDB attachment content - `Line` class: Now includes optional CouchDB metadata fields (doc_id, attachment_name, db_name) - `read_couchdb_partition()`: UDF function for processing partitions - `read_couchdb_rows()`: Non-distributed processing function - `read_couchdb_files_from_connection()`: Integration with CouchDBConnection

Purpose: Drop-in replacement for `file.py` when working with CouchDB attachments in PySpark. Uses the unified `Line` class with optional CouchDB fields.

1.2.2 2. examples/extract_taxa_from_couchdb.py

Complete working examples

Includes: - `extract_taxa_spark()`: Distributed Spark processing function - `extract_taxa_local()`: Local processing for testing/small datasets - `process_partition_to_taxa()`: Partition processing function - Three example scenarios: distributed, local, and filtering - Command-line interface

Purpose: Demonstrates how to use the CouchDB integration in production.

1.2.3 3. tests/test_couchdb_file.py

Comprehensive test suite

Test coverage: - CouchDBFile basic functionality - Line metadata preservation - Annotated content parsing - Page number track-

ing - Partition reading - Integration with `parse_annotated()` and `group_paragraphs()` - Full pipeline testing

Purpose: Ensures reliability and correctness of the CouchDB integration.

1.2.4 4. couchdb_file_README.md

Detailed module documentation

Contents: - Architecture overview - Class and function reference - Usage examples - Metadata tracking explanation - Best practices - Comparison with file.py

Purpose: Technical reference for developers using the module.

1.2.5 5. Updated EXTRACTING_TAXON_OBJECTS.md

Enhanced main documentation

New section added: - "Extracting from CouchDB (PySpark Distributed)" - Integration examples - Metadata preservation explanation - Benefits of CouchDB approach

Purpose: Complete guide covering both local file and CouchDB extraction methods.

1.3 Key Features

1.3.1 1. Metadata Preservation

When created from `CouchDBFile`, `Line` objects have optional CouchDB fields populated: - `doc_id`: CouchDB document ID (Optional[str]) - `attachment_name`: Attachment filename (Optional[str]) - `db_name`: Database name (ingest_db_name) (Optional[str]) - `filename`: Composite identifier "db_name/doc_id/attachment_name"

This metadata flows through the entire pipeline and appears in the final `Taxon` output. Regular file-based Lines have these fields as `None`.

1.3.2 2. Distributed Processing

Designed for PySpark's `mapPartitions()`: - One function call processes an entire partition - Minimizes serialization overhead - Scales to millions of documents - Compatible with existing Spark infrastructure

1.3.3 3. Drop-in Compatibility

Works seamlessly with existing code:

```
# Before: Local files
lines = read_files(file_paths)
paragraphs = parse_annotated(lines)
taxa = group_paragraphs(paragraphs)

# After: CouchDB (just change line reading)
lines = read_couchdb_partition(partition, db_name)
paragraphs = parse_annotated(lines)  # Same!
taxa = group_paragraphs(paragraphs)   # Same!
```

1.3.4 4. Efficient Integration with CouchDBConnection

Leverages existing `CouchDBConnection` class: - Uses `load_distributed()` for parallel attachment fetching - One connection per partition (not per document) - Automatic retry and error handling - Supports authentication

1.4 Architecture

1.4.1 Class Hierarchy

```
FileObject (Abstract Interface)
├── File (file.py)
│   └── read_line() → Line (CouchDB fields = None)
└── CouchDBFile (couchdb_file.py)
    └── read_line() → Line (CouchDB fields populated)

Line (line.py) - Unified class with optional CouchDB support
├── Standard fields: filename, line_number, page_number, etc.
└── Optional CouchDB fields: doc_id, attachment_name, db_name
```

1.4.2 Data Flow

```
CouchDB Database
  ↓
CouchDBConnection.load_distributed(spark, "*.txt.ann")
  ↓
DataFrame[doc_id, attachment_name, value]
  ↓
df.rdd.mapPartitions(lambda p: read_couchdb_partition(p, "mycobank"))
  ↓
Line objects (with CouchDB metadata fields populated)
```

```
  ↓
parse_annotated(lines)
  ↓
Paragraph objects (filename = "db_name/doc_id/attachment_name")
  ↓
group_paragraphs(paragraphs)
  ↓
Taxon objects (metadata preserved in dictionaries)
```

1.5 Usage Examples

1.5.1 Distributed Processing (Production)

```python
from pyspark.sql import SparkSession
from skol_classifier.couchdb_io import CouchDBConnection
from couchdb_file import read_couchdb_partition
from finder import parse_annotated, remove_interstitials
from taxon import group_paragraphs

def process_partition_to_taxa(partition, db_name):
    lines = read_couchdb_partition(partition, db_name)
    paragraphs = parse_annotated(lines)
    filtered = remove_interstitials(paragraphs)
    taxa = group_paragraphs(filtered)

    for taxon in taxa:
        for para_dict in taxon.dictionaries():
            # Extract metadata
            parts = para_dict['filename'].split('/', 2)
            if len(parts) == 3:
                para_dict['db_name'] = parts[0]
                para_dict['doc_id'] = parts[1]
                para_dict['attachment_name'] = parts[2]
            yield para_dict

# Setup Spark
spark = SparkSession.builder.appName("TaxonExtractor").getOrCreate()

# Load from CouchDB
conn = CouchDBConnection("http://localhost:5984", "mycobank", "user", "pass")
df = conn.load_distributed(spark, "*.txt.ann")

# Process in parallel
taxa_rdd = df.rdd.mapPartitions(
    lambda part: process_partition_to_taxa(part, "mycobank")
)
```

```
# Save results
taxa_df = taxa_rdd.toDF(schema)
taxa_df.write.parquet("output/taxa.parquet")
```

1.5.2 Local Processing (Testing/Small Datasets)

```
from couchdb_file import read_couchdb_files_from_connection
from skol_classifier.couchdb_io import CouchDBConnection

conn = CouchDBConnection("http://localhost:5984", "mycobank")
lines = read_couchdb_files_from_connection(conn, spark, "mycobank", "*.txt.ann")

paragraphs = parse_annotated(lines)
taxa = list(group_paragraphs(paragraphs))

for taxon in taxa:
    print(taxon.as_row())
```

1.5.3 Command-Line Usage

```
# Distributed mode
python examples/extract_taxa_from_couchdb.py \
    --mode distributed \
    --database mycobank_annotations \
    --db-name mycobank \
    --username admin \
    --password secret

# Local mode (for testing)
python examples/extract_taxa_from_couchdb.py \
    --mode local \
    --database mycobank_annotations \
    --db-name mycobank
```

1.6 Metadata Format

1.6.1 Composite Filename

CouchDB metadata is encoded in the `filename` property:

```
Format: db_name/doc_id/attachment_name
Example: mycobank/article_2023_001/fulltext.txt.ann
```

1.6.2 Parsing Metadata

```
parts = para_dict['filename'].split('/', 2)
db_name = parts[0]           # "mycobank"
doc_id = parts[1]            # "article_2023_001"
attachment_name = parts[2]   # "fulltext.txt.ann"
```

1.6.3 Output Schema

Recommended DataFrame schema for taxa:

```
schema = StructType([
    StructField("serial_number", StringType(), False),  # Taxon ID
    StructField("db_name", StringType(), True),         # Source database
    StructField("doc_id", StringType(), True),          # CouchDB doc ID
    StructField("attachment_name", StringType(), True), # Attachment name
    StructField("label", StringType(), False),          # Nomenclature/Descripti
    StructField("paragraph_number", StringType(), False),
    StructField("page_number", StringType(), False),
    StructField("body", StringType(), False),           # Full text
])
```

1.7 Benefits

1.7.1 1. Scalability

- Process millions of documents in parallel using Spark
- Efficient partition-level processing
- Linear scaling with cluster size

1.7.2 2. Traceability

- Full lineage from database to extracted taxa
- Track which documents produced which taxa
- Enable reproducibility and auditing

1.7.3 3. Flexibility

- Works with existing parser functions
- Compatible with local file processing
- Easy to switch between local and distributed modes

1.7.4 4. Efficiency

- One connection per partition (not per document)
- Minimal serialization overhead

- Reuses existing CouchDBConnection infrastructure

1.7.5 5. Maintainability

- Clean separation of concerns
- Well-tested components
- Comprehensive documentation

1.8 Testing

Run the test suite:

```
cd tests
python test_couchdb_file.py
```

Test coverage includes: - ✓ Basic CouchDBFile functionality - ✓ Line CouchDB metadata preservation - ✓ Annotated content parsing - ✓ Page number tracking - ✓ Partition reading - ✓ Integration with parse_annotated() - ✓ Full pipeline (Lines → Paragraphs → Taxa)

1.9 Migration Guide

1.9.1 From Local File Processing

Before:

```
from file import read_files
files = ['file1.txt.ann', 'file2.txt.ann']
lines = read_files(files)
paragraphs = parse_annotated(lines)
taxa = group_paragraphs(paragraphs)
```

After (Local CouchDB):

```
from couchdb_file import read_couchdb_files_from_connection
from skol_classifier.couchdb_io import CouchDBConnection

conn = CouchDBConnection("http://localhost:5984", "mycobank")
lines = read_couchdb_files_from_connection(conn, spark, "mycobank", "*.txt.ann")
paragraphs = parse_annotated(lines)  # Same!
taxa = group_paragraphs(paragraphs)   # Same!
```

After (Distributed CouchDB):

```
from couchdb_file import read_couchdb_partition

def process_partition(partition):
    lines = read_couchdb_partition(partition, "mycobank")
    paragraphs = parse_annotated(lines)
```

```
    return group_paragraphs(paragraphs)

conn = CouchDBConnection("http://localhost:5984", "mycobank")
df = conn.load_distributed(spark, "*.txt.ann")
taxa_rdd = df.rdd.mapPartitions(process_partition)
```

1.10 Documentation

- **Main Guide**: EXTRACTING_TAXON_OBJECTS.md
- **Module Reference**: couchdb_file_README.md
- **Examples**: examples/extract_taxa_from_couchdb.py
- **Tests**: tests/test_couchdb_file.py

1.11 Quick Reference

Task	Function	Use Case
Distributed processing	`read_couchdb_partitioned()`	Production, large datasets
Local processing	`read_couchdb_rows()`	Testing, small datasets
Integrated pipeline	`read_couchdb_files_from_connection()`	Quick setup, prototyping
Extract metadata	`filename.split('/', 2)`	Parse composite filename

1.12 Requirements

- PySpark
- couchdb Python library
- Existing modules: line.py, paragraph.py, finder.py, taxon.py
- skol_classifier.couchdb_io (for CouchDBConnection)

1.13 Summary

This CouchDB integration provides a complete, production-ready solution for extracting Taxon objects from annotated files stored in CouchDB. It maintains compatibility with existing code while adding powerful distributed processing capabilities and comprehensive metadata tracking.

The solution is: - **Scalable**: Handles millions of documents using Spark - **Traceable**: Preserves database metadata throughout - **Compatible**: Works with existing parser functions - **Efficient**: Optimized for

distributed processing - **Well-documented**: Comprehensive guides and examples - **Well-tested**: Extensive test coverage

Use `read_couchdb_partition()` as a UDF alternative to `read_files()` when processing CouchDB attachments in PySpark partitions, and enjoy the benefits of distributed processing while maintaining full traceability to the source database.