# MIDI Analysis Framework

The MIDI Analysis Framework was developed to compare real-world datasets before and after de-identification by the MIDI pipeline. This was necessary because the location of the PHI/PII in the real-world datasets is unknown. The framework was set up so that any two sets of DICOM images can be compared. The MIDI Analysis Framework only works for DICOM header information, not pixel information. [DCM4CHE](https://web.dcm4che.org/) is used within the MIDI Analysis Framework to extract the header information from the DICOM files. dcm4che is a Java-based library and toolset with functionalities for reading, writing, and manipulating DICOM data. Once the header information has been extracted into a CSV file, the CSV file is imported into BigQuery for further analysis.

Diagram, schematic

Description automatically generated

**Datasets**

The MIDI Phase 3 experiments for which the MIDI Analysis Framework was established compared four datasets.

|  |  |  |
| --- | --- | --- |
| Dataset | Provided by | Explanation |
| Source | UAMS | Real-world myeloma dataset containing PHI and PII |
| CTP | UAMS | Source dataset de-identified by the Clinical Trials Processor (CTP) |
| TCIA-curated | UAMS | CTP dataset subjected to manual review by TCIA curators |
| MIDI | Google/Deloitte | Dataset de-identified by the MIDI pipeline using either the source or the CTP dataset as input |

**Mapping files**

In addition to the datasets, UAMS provided a mapping file listing all original and updated UIDs in the source and the CTP datasets, including the SOP Instance UIDs of the source dataset and the SOP Instance UIDs of the CTP dataset. Google/Deloitte produced a mapping file as part of the MIDI pipeline deidentification process that linked the input dataset (source or CTP) with the output dataset (MIDI).

**Skillsets**

It is highly recommended that the developer using the analysis framework is knowledgeable of Java, SQL, DICOM, and the Google Cloud environment, including BigQuery.

# Downloading and Building the Project

## Code Location

<https://github.com/CBIIT/MIDI>

## Requirements

Java

Maven 3.8.5

## Building the Project

mvn clean compile assembly:single

## Build Product

A single jar is created in the target directory. This jar includes all dependencies. This jar is later renamed to midi-analysis.jar

Directory of C:\MIDI2\target

12/28/2022 05:44 AM <DIR> .

12/28/2022 05:44 AM <DIR> ..

12/28/2022 05:44 AM <DIR> archive-tmp

12/28/2022 05:44 AM <DIR> classes

12/28/2022 05:44 AM <DIR> generated-sources

12/28/2022 05:44 AM <DIR> maven-status

12/28/2022 05:44 AM 7,323,796 **midi-analysis-1.0-SNAPSHOT-jar-with-dependencies.jar**

# Running the Script to Build CVS files of DICOM Headers

# Using Google Buckets and Zip Files

This process aims to compare two image datasets. In most cases, this will be a dataset before and after de-identification. The datasets should be provided in tar-compressed (\*.tgz) format. A mapping file that links the UIDs of the input dataset's UIDs to the output dataset is required if the UIDs were updated during the de-identification process.

The jar file and the mapping file need to be copied to the machine that will run the script. The compressed image datasets stay in the Google bucket.  
  
Additional information: The DICOM files of the three UAMS datasets (source, CTP, TCIA-curated) were transferred to the Google buckets as compressed (e.g. midi\_3\_ctp\_dataset.tgz) files. Initially, the compressed files were expanded, but it turned out that running the comparison framework on the compressed files was the preferred method for running the script to create the CSV results files. This method of using the .tgz files is 10 times faster than mapping the buckets that had originally been attempted, and no expansion of the .tgz files is necessary.

## Run the Script Named RunProcess

java -classpath "/…/midi\_app/midi-analysis.jar" gov.nih.nci.midi.RunProcess tcia\_ctp tcia midi\_3\_ctp none /…/midi\_app/ nih-nci-cbiit-midi-dev2 midi\_3\_ctp\_dataset.tgz midibigqueryupload

**Script Parameters**

|  |  |
| --- | --- |
| **Definition** | **Example Parameter Value** |
| The name of the run, this will be used to name the output csv | tcia\_ctp |
| The component being tested {midi,tcia,ctp} | tcia |
| The bucket containing the tgz file | midi\_3\_ctp |
| The path to the mapping file, none means no mapping file is being used | none |
| The path to the output directory, this is a directory that is used to build the CSV file before it is transferred to the output bucket by the program | /…/midi\_app/ |
| The project name, this is used to provide authorization for the program to access the buckets | nih-nci-cbiit-midi-dev2 |
| The input zip file name | midi\_3\_ctp\_dataset.tgz |
| The output bucket | midibigqueryupload |

# Using Mapped Buckets and Expanded Zip Files

This is the **legacy method** for running the script to create the CSV output files. This method requires that the zip files are expanded, and buckets are mapped to directories, and it will run much slower than the direct-to-bucket method described in Section 2.1 of the document.

# Preparing the run environment

Use instance midi-linux-vm-analysis -1

Mapping to buckets

Output Bucket – There is a single output bucket that will hold the output csv files

Create the directory you want to use for output and map it

cd midi\_app/

mkdir output

gcsfuse midibigqueryupload /…/midi\_app/output

Create the directory you want to use for input and map it to the bucket where the DICOM files are using the switch --implicit-dirs to discover existing directories

mkdir input

cd input

mkdir 21\_patient

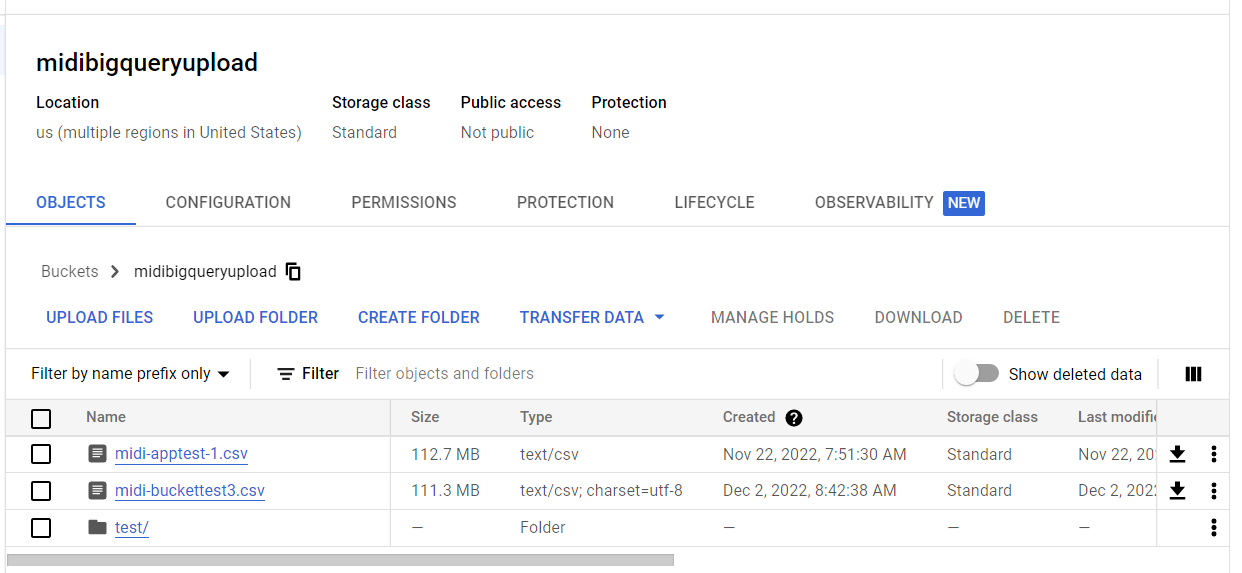
gcsfuse --implicit-dirs 21patients\_10-19 /…/midi\_app/input/21\_patient

## Run the Script Named RunProcess

java -classpath "midi-analysis.jar" gov.nih.nci.midi.RunProcess buckettest3 midi /…/input/21\_patient none /…/midi\_app/output

**Script Parameters**

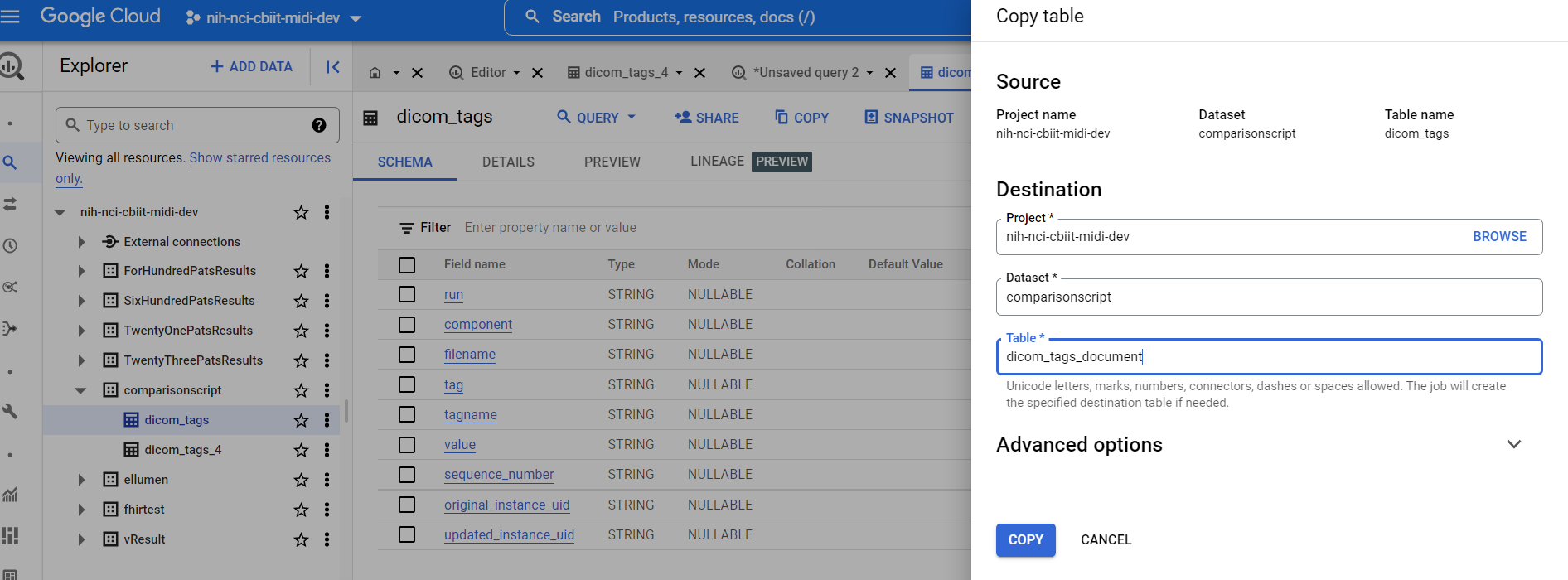
|  |  |
| --- | --- |
| Definition | Example Parameter Value |
| The name of the run, this will be used to name the output csv | buckettest3 |
| The component being tested {midi,tcia,ctp} | midi |
| The path to the input directory | /…/input/21\_patient |
| The path to the crosswalk file, none means no crosswalk | none |
| The path to the output directory | /…/midi\_app/output |

The component (midi) and the run (buckettest3) will be concatenated into the name of the CSV file to create a unique name, e.g., midi-buckettest3.csv, as shown in the screenshot.

# Load to BigQuery

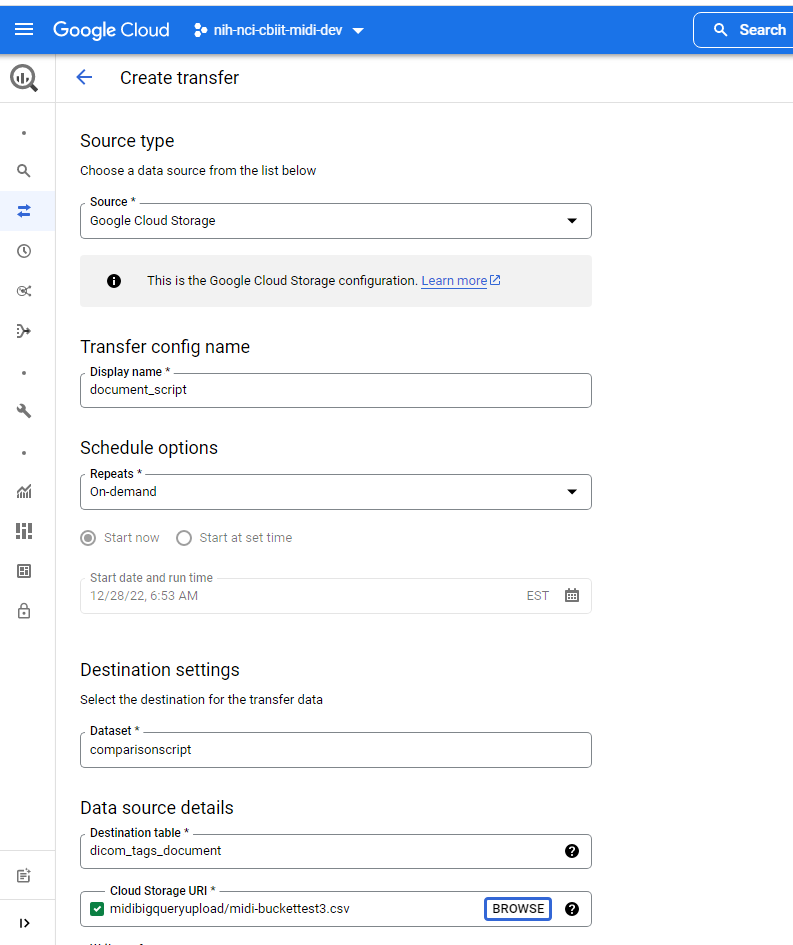
The CSV file created in step 2 contains the information extracted from the DICOM header of one dataset. The content of the CSV file will be loaded into a BigQuery table for further analysis.

## Create a new table for the load based on dicom\_tags



|  |  |  |
| --- | --- | --- |
| **Field name** | **Type** | **Explanation / Sample value** |
| Run | String | Name used in the parsing of the DICOM header information |
| Component | String | Component name used during the parsing of the DICOM header information |
| Filename | String | Path to the DICOM file |
| Tag | String | (0010,0010) |
| Tagname | String | PatientName |
| Value | String | Jane Doe |
| Sequence number | String | Level of the tag within a sequence |
| Original\_instance\_UID | String | SOP instance UID of the original dataset, e.g., dataset before deidentification |
| Updated\_instance\_UID | String | SOP instance UID of the updated dataset e.g., dataset after deidentification |

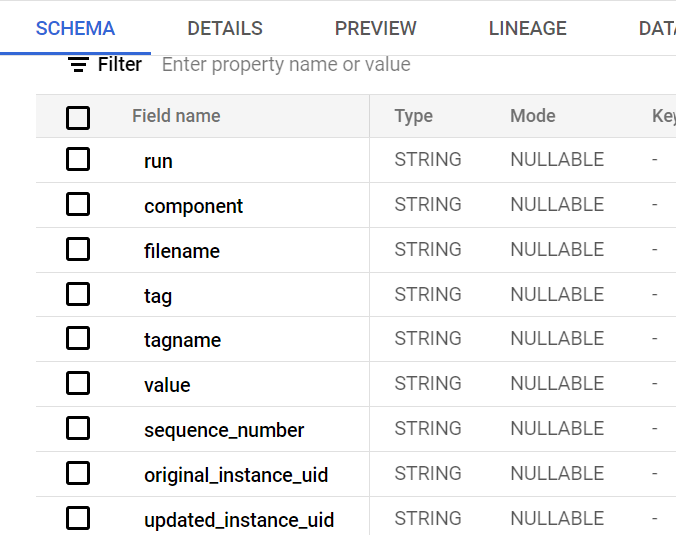
## Load the data from the output CSV file in the bucket by creating and running a data transfer



You can now use SQL to compare runs by joining the tables on the tags. Once you complete the analysis, you delete the table

# Analyzing Data in Big Query

Once the data is in Big Query, the analysis can begin. The design of the tables is shown below.



|  |  |  |
| --- | --- | --- |
| **Field name** | **Type** | **Explanation / Sample value** |
| Run | String | Name used in the parsing of the DICOM header information |
| Component | String | Component name used during the parsing of the DICOM header information |
| Filename | String | Path to the DICOM file |
| Tag | String | (0010,0010) |
| Tagname | String | PatientName |
| Value | String | Jane Doe |
| Sequence number | String | Level of the tag within a sequence |
| Original\_instance\_UID | String | SOP instance UID of the original dataset, e.g., dataset before deidentification |
| Updated\_instance\_UID | String | SOP instance UID of the updated dataset e.g., dataset after deidentification |

## The Analysis approach

Three datasets are compared during the analysis.

Diagram

Description automatically generated

These are the setups for the MIDI Phase 3 experiment.

Diagram

Description automatically generatedDiagram

Description automatically generated

The following findings are anticipated:

1. **No difference** 
   1. (source = output 1 = output 2)
      1. No difference or functionally the same (e.g. null data elements are the same as missing data elements)
      2. Problem for type 2 data elements? They are mandatory but the value can be of zero length.
2. **Difference 1** 
   1. (source ≠ output 1)
   2. (source = output 2)
3. **Difference 2** 
   1. (source = output 1)
   2. (source ≠ output 2)
4. **Difference both** 
   1. (source ≠ output 1 ≠ output 2)
5. **Difference equivalent** 
   1. (source ≠ output 1)
   2. (source ≠ output 2)
   3. (output 1= output 2)

## Tracking missing files

For convenience, we may need a column denoting if the DICOM file is in the dataset to track missing DICOM files on various runs.   
For example, the CTP dataset is compared to the TCIA-curated dataset, and later, the Source images are compared to the TCIA-curated images.

For the table created from CTP, we add a [in\_curated\_doc] column. It is set to Y when the tag is in a DICOM file in the TCIA-curated data set.

For the table created from the source dataset, we add a column [in\_curated\_doc] as well as a column [in\_ctp\_docs], which is set to Y when the tag is in a document that is in the CTP data set. Once the fields are set, we can compare sets of documents where we know the document is in both sets by adding the condition as with [in\_curated\_doc]=’Y’ to compare CTP to TCIA curated.

Below is an example of populating this column:

update `nih-nci-cbiit-midi-dev2.comparisonscript.tcia-tcia\_ctp` run1

 set run1.in\_curated\_doc = 'Y'

where run1.original\_instance\_uid in

(select distinct run2.original\_instance\_uid

from `nih-nci-cbiit-midi-dev2.comparisonscript.tcia-tcia\_data` run2)

## Queries to compare runs

Below is a query to compare CTP header information to TCIA-curated header information. Note that by using a full join, we can see when tags are not in a dataset by looking for nulls

SELECT run1.original\_instance\_uid,

run1.filename,

run1.tag,

run1.value,

run2.original\_instance\_uid,

run2.filename,

run2.tag,

run2.value

FROM `nih-nci-cbiit-midi-dev2.comparisonscript.tcia-tcia\_ctp` run1

full join `nih-nci-cbiit-midi-dev2.comparisonscript.tcia-tcia\_data` run2

on run1.original\_instance\_uid=run2.original\_instance\_uid

and run1.tag=run2.tag

and run1.sequence\_number=run2.sequence\_number

where (run1.tag is null

or run2.tag is null

or run2.value is distinct from run1.value)

and run1.in\_curated\_doc='Y'

## Tables for MIDI3

Several tables were created to support MIDI3; all have been truncated to protect PHI. The table structures are captured in the GitHub repository in the SQL directory in a filed name MIDI Tables.

* nih-nci-cbiit-midi-dev2.comparisonscript.midi-tcia\_ctp – The MIDI Pipeline output for the CTP file set.
* nih-nci-cbiit-midi-dev2.comparisonscript.midi-tcia\_source – The MIDI Pipeline output for the Source file set.
* nih-nci-cbiit-midi-dev2.comparisonscript.tcia-tcia\_source – The Source file set.
* nih-nci-cbiit-midi-dev2.comparisonscript.tcia-tcia\_ctp – The CTP Source file set.
* nih-nci-cbiit-midi-dev2.comparisonscript.tcia-tcia\_data -– The TCIA-curated file set.

The SQL to create the tables is stored in the [MIDI GitHub](https://github.com/CBIIT/MIDI/blob/main/sql/MIDI%20Tables.txt) repository's SQL directory.

## Queries for MIDI3

The queries used to analyze the MIDI data tables. Note that the Filenames queries need to be output to BigQuery tables as they are too large to export. The source SQL of these queries is in the [GitHub repository under SQL/Queries](https://github.com/CBIIT/MIDI/tree/main/sql/Queries)

* Count Images in SOURCE not in MIDI.sql
* Filenames MIDI Source No UIDS Tags not null in MIDI dataset not in TCIA with Source Values.sql
* Filenames MIDI Source No UIDs Updated tags in MIDI not updated in TCIA.sql
* Filenames MIDI Source No UIDs Updated tags in MIDI vs Update tags in TCIA.sql
* Filenames MIDI Source No UIDs Updated tags in TCIA not updated in MIDI.sql
* MIDI CTP Images in Source not in MIDI.sql
* MIDI CTP Images in Source not in TCIA.sql
* MIDI CTP Tags not null in MIDI dataset not in TCIA with Source value.sql
* MIDI CTP Updated tags in MIDI from source not updated in TCIA including values.sql
* MIDI CTP Updated tags in MIDI from source vs Update tags in TCIA from source including values.sql
* MIDI CTP Updated tags in TCIA from source not updated in MIDI including values.sql
* MIDI Not Updated UIDS.sql
* MIDI Source No UIDs Tags not null in MIDI dataset not in TCIA with Source value.sql
* MIDI Source Duplicates.sql
* MIDI Source Images in Source not in CTP.sql
* MIDI Source Images in Source not in MIDI.sql
* MIDI Source Images in Source not in TCIA curated data.sql
* MIDI Source Images in Source not in TCIA.sql
* MIDI Source No UIDS Tags not null in MIDI dataset not in TCIA with Source Values (1).sql
* MIDI Source No UIDS Tags not null in MIDI dataset not in TCIA with Source Values.sql
* MIDI Source No UIDs Updated tags in MIDI not updated in TCIA.sql
* MIDI Source No UIDs Updated tags in MIDI vs Update tags in TCIA from source including values.sql
* MIDI Source No UIDs Updated tags in TCIA not updated in MIDI including values.sql
* MIDI Source Update In Curated for MIDI.sql
* MIDI Source Updated tags in MIDI from source not updated in TCIA including values.sql
* MIDI Source Updated tags in MIDI from source vs Update tags in TCIA from source including values (1).sql
* MIDI Source Updated tags in MIDI from source vs Update tags in TCIA from source including values.sql
* MIDI Source Updated tags in TCIA from source not updated in MIDI including values.sql
* MIDI Source Updated tags not null in MIDI from source not updated in TCIA including values.sql
* TCIA Curated not in Source.sql
* TCIA Duplicates.sql
* TCIA MIDI Duplicates.sql
* TCIA Source Duplicates.sql