geobeam

Release 0.4.6

traviswebb@google.com

Jun 07, 2022

CONTENTS:

CHAPTER

ONE

MODULE DOCUMENTATION

1.1 geobeam package

1.1.1 Subpackages

1.1.1.1 geobeam.examples package

Submodules

geobeam.examples.geodatabase_frd module

Loads FRD (Flood Risk Database) layers into Bigquery using the *FILE_LOADS* insertion method. *FILE_LOADS* should be used when individual geometries can be large and complex to avoid the size limits of the *STREAMING_INSERTS* method. See: https://cloud.google.com/bigquery/quotas#streaming_inserts.

`geobeam.examples.geodatabase_frd.`**`run`**`(`*pipeline_args*, *known_args*`)`

Run the pipeline

geobeam.examples.geotiff_dem module

Example pipeline that loads a DEM (digital elevation model) raster into Bigquery.

`geobeam.examples.geotiff_dem.`**`elev_to_centimeters`**`(`*element*`)`

Convert the floating-point meters into rounded centimeters to store as INT64 in order to support clustering on this value column (elev).

`geobeam.examples.geotiff_dem.`**`run`**`(`*pipeline_args*, *known_args*`)`

Run the pipeline. Invoked by the Beam runner.

geobeam.examples.geotiff_soilgrid module

Example pipeline that loads a Soil Grid raster that contains groundwater saturation values.

`geobeam.examples.geotiff_soilgrid.`**`run`**`(`*pipeline_args*, *known_args*`)`

Run the pipeline. Invoked by the Beam runner.

geobeam.examples.shapefile_nfhl module

Example pipeline that loads the NFHL (National Flood Hazard Layer) into BigQuery.

geobeam.examples.shapefile_nfhl.**run**(*pipeline_args*, *known_args*)

Invoked by the Beam runner

Module contents

A set of examples that demonstrate *geobeam* functionality.

Usage: python -m geobeam.examples.<example> –runner=PortableRunner ... args

1.1.2 Submodules

1.1.3 geobeam.fn module

Beam functions, transforms, and filters that can be used to process geometries in your pipeline

geobeam.fn.**filter_invalid**(*element*)

Use with fn.make_valid to filter out geometries that are invalid, empty, or are out of bounds.

Example: .. code-block:: python

p | beam.Map(geobeam.fn.make_valid)

beam.Map(geobeam.fn.filter_invalid)

geobeam.fn.**format_record**(*element*, *band_column=None*, *band_type='int'*)

Format the tuple received from the geobeam file source into a record that can be inserted into BigQuery. If using a raster source, the bands and band_column will be combined.

Args:

band_column (str, optional): the name of the raster band column band_type (str, optional): Default to int. The data type of the

raster band column to store in the database.

Example: .. code-block:: python

vector p | beam.Map(geobeam.fn.format_record)

raster p | beam.Map(geobeam.fn.format_record,

band_column='elev', band_type=float)

geobeam.fn.**make_valid**(*element*, *drop_z=True*)

Attempt to make a geometry valid. Returns *None* if the geometry cannot be made valid.

Example: .. code-block:: python

p | beam.Map(geobeam.fn.make_valid)

beam.Map(geobeam.fn.filter_invalid)

geobeam.fn.**trim_polygons**(*element*, *d=1e-07*, *cf=1.2*)

Remove extraneous artifacts, tails, etc. from otherwise valid polygons

Args:

d (float, optional): trim distance cf (float, optional): corrective factor

Exmaple: .. code-block:: python

p | beam.Map(geobeam.fn.trim_polygons, d=0.00001, cf=1.2

1.1.4 geobeam.io module

This package contains Apache Beam I/O connectors for reading from spatial data files.

class geobeam.io.**ESRIServerSource**(*file_pattern*, *skip_reproject=False*, *in_epsg=None*, *in_proj=None*, ***kwargs*)

Bases: FileBasedSource

A Beam FileBasedSource for reading layers from an ESRI ArcGIS Server.

The given file(s) should be a link to a specific layer from the ArcGIS Server REST API (ex. https://services.arcgis.com/P3ePLMYs2RVChkJx/arcgis/rest/services/USA_States_Generalized/FeatureServer/0)

p | beam.io.Read(ESRIServerSource(file_pattern))

beam.Map(print)

Args:

skip_reproject (bool, optional): Defaults to *False*. True to return
geom in its original projection.

in_epsg (int, optional): override the source projection with an EPSG
code.

in_proj (str, optional): override the source projection with a
PROJ4 string.

Yields:

generator of (*props*, *geom*) tuples. *props* is a *dict* containing all of the feature properties. *geom* is the geometry.

read_records(*file_name*, *range_tracker*)

Returns a generator of records created by reading file 'file_name'.

Args:

file_name: a `string` that gives the name of the file to be read. Method
`FileBasedSource.open_file()` must be used to open the file and create a seekable file object.

offset_range_tracker: a object of type `OffsetRangeTracker`. This
defines the byte range of the file that should be read. See documentation in `iobase.BoundedSource.read()` for more information on reading records while complying to the range defined by a given `RangeTracker`.

Returns:

an iterator that gives the records read from the given file.

class geobeam.io.**GeoJSONSource**(*file_pattern*, *skip_reproject=False*, *in_epsg=None*, *in_proj=None*, ***kwargs*)

Bases: FileBasedSource

A Beam FileBasedSource for reading GeoJSON Files.

The given file(s) should be a .geojson file.

p | beam.io.Read(GeoJSONSource(file_pattern))

beam.Map(print)

Args:

skip_reproject (bool, optional): Defaults to *False*. True to return
geom in its original projection.

in_epsg (int, optional): override the source projection with an EPSG
code.

in_proj (str, optional): override the source projection with a
PROJ4 string.

Yields:
generator of (*props*, *geom*) tuples. *props* is a *dict* containing all of the feature properties. *geom* is the geometry.

read_records(*file_name*, *range_tracker*)

Returns a generator of records created by reading file 'file_name'.

Args:

file_name: a string that gives the name of the file to be read. Method
FileBasedSource.open_file() must be used to open the file and create a seekable file object.

offset_range_tracker: a object of type OffsetRangeTracker. This
defines the byte range of the file that should be read. See documentation in iobase.BoundedSource.read() for more information on reading records while complying to the range defined by a given RangeTracker.

Returns:
an iterator that gives the records read from the given file.

class geobeam.io.**GeodatabaseSource**(*file_pattern*, *gdb_name=None*, *layer_name=None*, *in_epsg=None*, *in_proj=None*, *skip_reproject=False*, ***kwargs*)

Bases: FileBasedSource

A Beam FileBasedSource for reading geodatabases.

The given file(s) should be a zip archive containing .gdb geodatabase directory.

p | beam.io.Read(GeodatabaseSource(file_pattern))

beam.Map(print)

Args:

gdb_name (str): Required. the name of the .gdb directory in the archive,
e.g. *FRD_510104_Coastal_GeoDatabase_20160708.gdb*

layer_name (str): Required. the name of the layer you want to read from
the gdb, e.g. *S_CSLF_Ar*

skip_reproject (bool, optional): Defaults to *False*. True to return
geom in its original projection.

in_epsg (int, optional): override the source projection with an EPSG
code.

in_proj (str, optional): override the source projection with a
PROJ4 string.

Yields:
generator of (*props*, *geom*) tuples. *props* is a *dict* containing all of the feature properties. *geom* is the geometry.

`read_records`(*file_name*, *range_tracker*)

Returns a generator of records created by reading file 'file_name'.

Args:

file_name: a `string` that gives the name of the file to be read. Method
`FileBasedSource.open_file()` must be used to open the file and create a seekable file object.

offset_range_tracker: a object of type `OffsetRangeTracker`. This
defines the byte range of the file that should be read. See documentation in `iobase.BoundedSource.read()` for more information on reading records while complying to the range defined by a given `RangeTracker`.

Returns:
an iterator that gives the records read from the given file.

`class geobeam.io.GeotiffSource`(*file_pattern*, *band_number=1*, *include_nodata=False*, *skip_reproject=False*, *centroid_only=False*, *in_epsg=None*, *in_proj=None*, *merge_blocks=32*, ***kwargs*)

Bases: `FileBasedSource`

A Beam FileBasedSource for reading Geotiff files.

The Geotiff is read in blocks and each block is polygonized. Each polygon is returned as a (*value*, *geom*) tuple, where *value* is the band value of the polygonized pixels, and *geom* is the Polygon (or Point if centroid_only is True) geometry that corresponds to the *value*. The raster is stored in RAM, so make sure you specify a machine_type with enough RAM to hold the entire raster image.

p | beam.io.Read(GeotiffSource(file_pattern))

beam.Map(print)

Args:
file_pattern (str): required, passed to FileBasedSource. band_number (int, optional): Defaults to *1*. the band(s) to read from

the raster.

include_nodata (bool, optional): Defaults to *False*. False to ignore
nodata values in the raster band; True to include them.

centroid_only (bool, optional): Defaults to *False*. True to set geom
to the centroid Point, False to include the entire Polygon. Do not use with the merge_blocks option.

skip_reproject (bool, optional): Defaults to *False*. True to return
geom in its original projection. This can be useful if your data is in a bespoke CRS that requires a custom reprojection, or if you want to join/clip with other spatial data in the same projection. Note: you will need to manually reproject all geometries to EPSG:4326 in order to store it in BigQuery.

in_epsg (int, optional): override the source projection with an EPSG
code.

in_proj (str, optional): override the source projection with a
PROJ4 string.

merge_blocks (int, optional): Defaults to *32*. Number of windows
to combine during polygonization. Setting this to a larger number will result in fewer file reads and possible improved overall performance. Setting this value too high (>100) may cause file read issues and worker timeouts. Set to a smaller number if your raster blocks are large (256x256 or larger).

Yields:
generator of (*value*, *geom*) tuples. The data type of *value* is determined by the raster band it came from.

`read_records`(*file_name*, *range_tracker*)

Returns a generator of records created by reading file 'file_name'.

Args:

file_name: a `string` that gives the name of the file to be read. Method
`FileBasedSource.open_file()` must be used to open the file and create a seekable file object.

offset_range_tracker: a object of type `OffsetRangeTracker`. This
defines the byte range of the file that should be read. See documentation in `iobase.BoundedSource.read()` for more information on reading records while complying to the range defined by a given `RangeTracker`.

Returns:
an iterator that gives the records read from the given file.

`class geobeam.io.`**`ShapefileSource`**(*file_pattern*, *layer_name=None*, *skip_reproject=False*, *in_epsg=None*, *in_proj=None*, ***kwargs*)

Bases: `FileBasedSource`

A Beam FileBasedSource for reading shapefiles.

The given file(s) should be a zip archive containing the .shp file alongside the .dbf and .prj files.

p | beam.io.Read(ShapefileSource(file_pattern))

beam.Map(print)

Args:

layer_name (str, optional): the name of the layer you want to read.
Required the zipfile contains multiple layers.

skip_reproject (bool, optional): Defaults to *False*. True to return
geom in its original projection.

in_epsg (int, optional): override the source projection with an EPSG
code.

in_proj (str, optional): override the source projection with a
PROJ4 string.

Yields:
generator of (*props*, *geom*) tuples. *props* is a *dict* containing all of the feature properties. *geom* is the geometry.

`read_records`(*file_name*, *range_tracker*)

Returns a generator of records created by reading file 'file_name'.

Args:

file_name: a `string` that gives the name of the file to be read. Method
`FileBasedSource.open_file()` must be used to open the file and create a seekable file object.

offset_range_tracker: a object of type `OffsetRangeTracker`. This
defines the byte range of the file that should be read. See documentation in `iobase.BoundedSource.read()` for more information on reading records while complying to the range defined by a given `RangeTracker`.

Returns:
an iterator that gives the records read from the given file.

1.1.5 geobeam.util module

This module contains utility functions that make working with geosaptial data in Google Cloud easier.

`geobeam.util.`**`get_bigquery_raster_schema`**(*band_column='value'*, *band_type='INT64'*)

Generate Bigquery table schema for a raster

`geobeam.util.`**`get_bigquery_schema`**(*filepath*, *layer_name=None*, *gdb_name=None*)

Generate a Bigquery table schema from a geospatial file

python -m geobeam.util get_bigquery_schema ... args

Args:
filepath (str): full path to the input file layer_name (str, optional): name of the layer, if file contains

multiple layers

Returns:
dict: the schema, convertable to json by json.dumps(schema, indent=2)

`geobeam.util.`**`get_bigquery_schema_dataflow`**(*filepath*, *layer_name=None*, *gdb_name=None*)

Generate a Bigquery table schema from a geospatial file hosted on a Google Cloud Storage bucket

from apache_beam.io.gcp.bigquery_tools import parse_table_schema_from_json

table_schema = parse_table_schema_from_json(get_bigquery_schema_dataflow(known_args.gcs_url, known_args.layer_name))

Args:
filepath (str): full path to the input file hosted on Google Cloud Storage layer_name (str, optional): name of the layer, if file contains

multiple layers

Returns:
JSON: the schema in JSON that can be passed to the schema argument in WriteToBigQuery. Must use the parse_table_schema_from_json() from apache_beam.io.gcp.bigquery_tools

1.1.6 Module contents

geobeam root namespace.

CHAPTER

TWO

README

geobeam adds GIS capabilities to your Apache Beam pipelines.

2.1 What does geobeam do?

`geobeam` enables you to ingest and analyze massive amounts of geospatial data in parallel using Dataflow. geobeam provides a set of FileBasedSource classes that make it easy to read, process, and write geospatial data, and provides a set of helpful Apache Beam transforms and utilities that make it easier to process GIS data in your Dataflow pipelines.

See the Full Documentation for complete API specification.

2.1.1 Requirements

- Apache Beam 2.27+
- Python 3.7+

  Note: Make sure the Python version used to run the pipeline matches the version in the built container.

2.1.2 Supported input types

File format	Data type	Geobeam class
`tiff`	raster	`GeotiffSource`
`shp`	vector	`ShapefileSource`
`gdb`	vector	`GeodatabaseSource`
`json`	vector	`GeoJSONSource`
URL	vector	`ESRIServerSource`

2.1.3 Included libraries

`geobeam` includes several python modules that allow you to perform a wide variety of operations and analyses on your geospatial data.

Module	Version	Description
gdal	3.2.1	python bindings for GDAL
rasterio	1.1.8	reads and writes geospatial raster data
fiona	1.8.18	reads and writes geospatial vector data
shapely	1.7.1	manipulation and analysis of geometric objects in the cartesian plane
esridump	1.10.1	read layer from ESRI server

2.2 How to Use

2.2.1 1. Install the module

```
pip install geobeam
```

2.2.2 2. Write your pipeline

Write a normal Apache Beam pipeline using one of geobeams file sources. See ``*geobeam/examples*` <https://github.com/GoogleCloudPlatform/dataflow-geobeam/tree/main/geobeam/examples>`_ for inspiration.

2.2.3 3. Run

2.2.3.1 Run locally

```
python -m geobeam.examples.geotiff_dem \
  --gcs_url gs://geobeam/examples/dem-clipped-test.tif \
  --dataset=examples \
  --table=dem \
  --band_column=elev \
  --centroid_only=true \
  --runner=DirectRunner \
  --temp_location <temp gs://> \
  --project <project_id>
```

You can also run "locally" in Cloud Shell using the ``*py-37*` container variants <https://github.com/GoogleCloudPlatform/dataflow-geobeam#note-on-python-versions>`_

Note: Some of the provided examples may take a very long time to run locally...

2.2.3.2 Run in Dataflow

Write a Dockerfile

This will run in Dataflow as a custom container based on the ``*dataflow-geobeam/base*` <Dockerfile>`_ image. See [geobeam/examples/Dockerfile] for an example that installed the latest geobeam from source.

```
FROM gcr.io/dataflow-geobeam/base
# FROM gcr.io/dataflow-geobeam/base-py37
```

(continues on next page)

(continued from previous page)

```
RUN pip install geobeam

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
```

```
# build locally with docker
docker build -t gcr.io/<project_id>/example
docker push gcr.io/<project_id>/example

# or build with Cloud Build
gcloud builds submit --tag gcr.io/<project_id>/<name> --timeout=3600s --machine-type=n1-
↪highcpu-8
```

2.2.3.3 Start the Dataflow job

Note on Python versions

If you are starting a Dataflow job on a machine running Python 3.7, you must use the images suffixed with **py-37**. (Cloud Shell runs Python 3.7 by default, as of Feb 2021). A separate version of the base image is built for Python 3.7, and is available at `gcr.io/dataflow-geobeam/base-py37`. The Python 3.7-compatible examples image is similarly-named `gcr.io/dataflow-geobeam/example-py37`

```
# run the geotiff_soilgrid example in dataflow
python -m geobeam.examples.geotiff_soilgrid \
  --gcs_url gs://geobeam/examples/AWCh3_M_sl1_250m_ll.tif \
  --dataset=examples \
  --table=soilgrid \
  --band_column=h3 \
  --runner=DataflowRunner \
  --worker_harness_container_image=gcr.io/dataflow-geobeam/example \
  --experiment=use_runner_v2 \
  --temp_location=<temp bucket> \
  --service_account_email <service account> \
  --region us-central1 \
  --max_num_workers 2 \
  --machine_type c2-standard-30 \
  --merge_blocks 64
```

2.3 Examples

```
def run(options):
  from geobeam.io import GeotiffSource
  from geobeam.fn import format_record

  with beam.Pipeline(options) as p:
    (p  | 'ReadRaster' >> beam.io.Read(GeotiffSource(gcs_url))
```

(continues on next page)

(continued from previous page)

```
        | 'FormatRecord' >> beam.Map(format_record, 'elev', 'float')
        | 'WriteToBigquery' >> beam.io.WriteToBigQuery('geo.dem'))
```

```
def run(options):
  from geobeam.io import ShapefileSource
  from geobeam.fn import make_valid, filter_invalid, format_record

  with beam.Pipeline(options) as p:
    (p  | 'ReadShapefile' >> beam.io.Read(ShapefileSource(gcs_url))
        | 'Validate' >> beam.Map(make_valid)
        | 'FilterInvalid' >> beam.Filter(filter_invalid)
        | 'FormatRecord' >> beam.Map(format_record)
        | 'WriteToBigquery' >> beam.io.WriteToBigQuery('geo.parcel'))
```

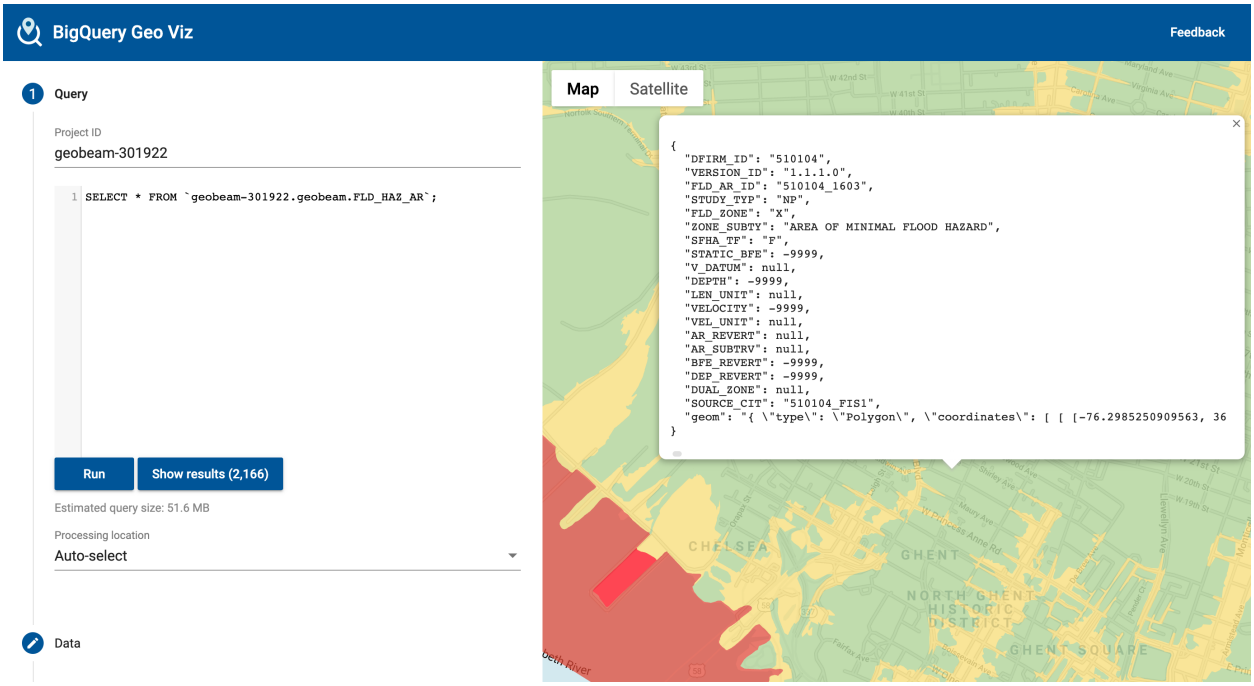
See `geobeam/examples/` for complete examples.

A number of example pipelines are available in the `geobeam/examples/` folder. To run them in your Google Cloud project, run the included terraform file to set up the Bigquery dataset and tables used by the example pipelines.

Open up Bigquery GeoViz to visualize your data.

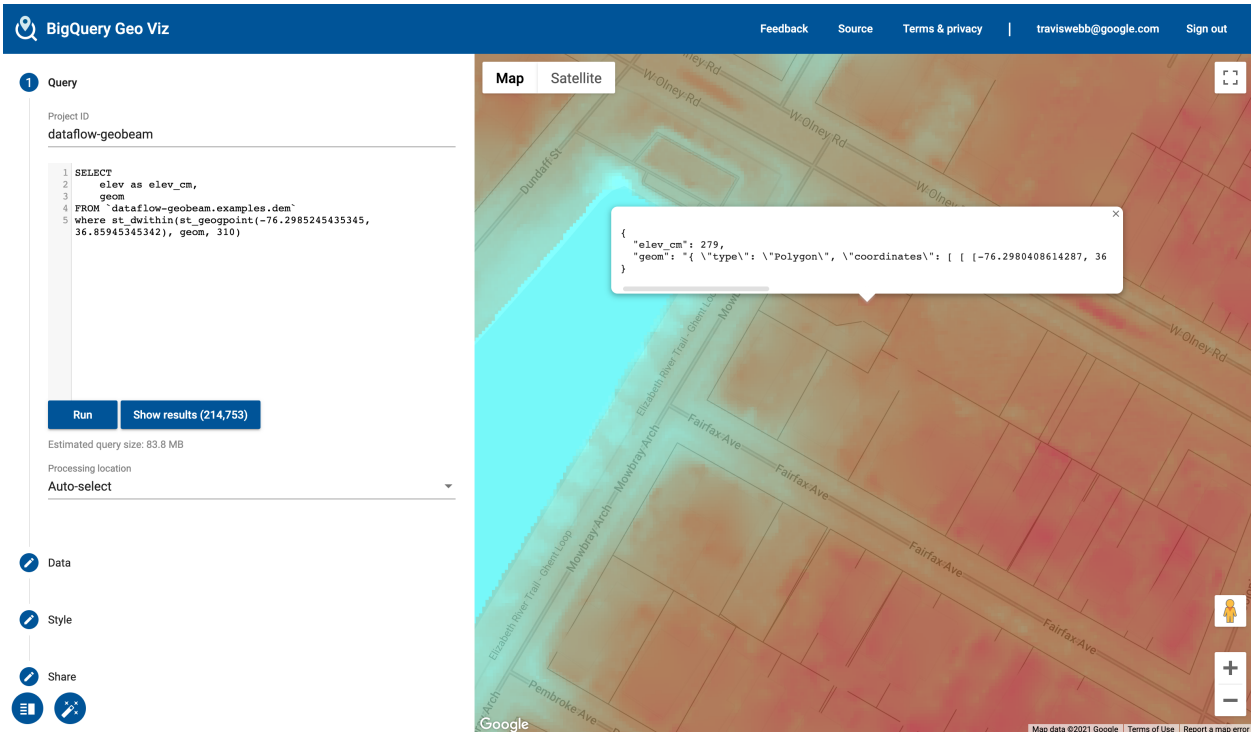
2.3.1 Shapefile Example

The National Flood Hazard Layer loaded from a shapefile. Example pipeline at ``*geobeam/examples/shapefile_nfhl.py*`` <https://github.com/GoogleCloudPlatform/dataflow-geobeam/blob/main/geobeam/examples/shapefile_nfhl.py>`_



2.3.2 Raster Example

The Digital Elevation Model is a high-resolution model of elevation measurements at 1-meter resolution. (Values converted to centimeters). Example pipeline: ``*geobeam/examples/geotiff_dem.py*` <https://github.com/GoogleCloudPlatform/dataflow-geobeam/blob/main/geobeam/examples/geotiff_dem.py>`_.



2.4 Included Transforms

The `geobeam.fn` module includes several Beam Transforms that you can use in your pipelines.

Module	Description
`geobeam.fn.make_valid`	Attempt to make all geometries valid.
`geobeam.fn.filter_invalid`	Filter out invalid geometries that cannot be made valid
`geobeam.fn.format_record`	Format the (props, geom) tuple received from a FileSource into a `dict` that can be inserted into the destination table

2.5 Execution parameters

Each FileSource accepts several parameters that you can use to configure how your data is loaded and processed. These can be parsed as pipeline arguments and passed into the respective FileSources as seen in the examples pipelines.

Parameter	Input type	Description	Default	Required?
`skip_reproject`	All	True to skip reprojection during read	`False`	No
`in_epsg`	All	An EPSG integer to override the input source CRS to reproject from		No
`in_proj`	All	A PROJ string to override the input source CRS		No
`band_number`	Raster	The raster band to read from	`1`	No
`include_nodata`	Raster	True to include `nodata` values	`False`	No
`centroid_only`	Raster	True to only read pixel centroids	`False`	No
`merge_blocks`	Raster	Number of block windows to combine during read. Larger values will generate larger, better-connected polygons.		No
`layer_name`	Vector	Name of layer to read		Yes, for shapefiles
`gdb_name`	Vector	Name of geodatabase directory in a gdb zip archive		Yes, for GDB files

2.6 License

This is not an officially supported Google product, though support will be provided on a best-effort basis.

```
Copyright 2021 Google LLC

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER

THREE

EXAMPLES

CHAPTER

FOUR

GEOBEAM EXAMPLES

4.1 shapefile_parcel

Load a shapefile of county parcels into Bigquery

4.1.1 Run locally

```
python -m geobeam.examples.shapefile_parcel \
  --runner DirectRunner \
  --project <your project>
  --temp_location gs://geobeam-pipeline-tmp \
  --gcs_url gs://geobeam/examples/ghent-parcels-shp.zip \
  --layer_name Property_Information \
  --dataset examples \
  --table parcel
```

4.1.2 Run in Dataflow

```
python -m geobeam.examples.shapefile_parcel \
  --runner DataflowRunner \
  --worker_harness_container_image gcr.io/dataflow-geobeam/example \
  --experiment use_runner_v2 \
  --project=dataflow-geobeam \
  --temp_location <your temp bucket>
  --service_account_email <your service account>
  --region us-central1
  --gcs_url gs://geobeam/examples/ghent-parcels-shp.zip \
  --layer_name Property_Information \
  --dataset examples \
  --table parcel
```

4.2 geodatabase_frd

4.2.1 Run locally

```
python -m geobeam.examples.geodatabase_frd \
  --runner DirectRunner \
  --project <your project> \
  --temp_location <your temp bucket> \
  --gcs_url gs://geobeam/examples/FRD_510104_Coastal_GeoDatabase_20160708.zip \
  --dataset examples \
  --table CSLF_Ar \
  --gdb_name FRD_510104_Coastal_GeoDatabase_20160708.gdb \
  --layer_name S_CSLF_Ar
```

4.2.2 Run in Dataflow

```
python -m geobeam.examples.geodatabase_frd \
  --project <your project> \
  --runner DataflowRunner \
  --worker_harness_container_image gcr.io/dataflow-geobeam/example \
  --experiment use_runner_v2 \
  --temp_location <your temp bucket> \
  --service_account_email <your service account> \
  --region us-central1 \
  --gcs_url gs://geobeam/examples/FRD_510104_Coastal_GeoDatabase_20160708.zip \
  --gdb_name FRD_510104_Coastal_GeoDatabase_20160708.gdb \
  --layer_name S_CSLF_Ar \
  --dataset examples \
  --table CSLF_Ar
```

4.3 geotiff_dem

Load a Digital Elevation Model (DEM) raster into Bigquery

4.3.1 Run Locally

```
python -m geobeam.examples.geotiff_dem \
  --runner DirectRunner \
  --temp_location <your temp bucket> \
  --project <your project> \
  --gcs_url gs://geobeam/examples/ghent-dem-1m.tif \
  --band_column elev \
  --centroid_only true \
  --skip_nodata true
```

4.3.2 Run in Dataflow

```
python -m geobeam.examples.geotiff_dem \
  --runner DataflowRunner \
  --worker_harness_container_image gcr.io/dataflow-geobeam/example \
  --experiment use_runner_v2 \
  --project dataflow-geobeam \
  --temp_location gs://geobeam-pipeline-tmp/ \
  --service_account_email dataflow-runner@dataflow-geobeam.iam.gserviceaccount.com \
  --region us-central1 \
  --gcs_url gs://geobeam/examples/dem-clipped-test.tif \
  --dataset examples \
  --table dem \
  --schema 'elev:INT64,geom:GEOGRAPHY'
  --band_column elev \
  --max_num_workers 3 \
  --machine_type c2-standard-30 \
  --merge_blocks 80 \
  --centroid_only true \
```

4.4 geotiff_soilgrid

4.4.1 Run Locally

```
python -m geobeam.examples.geotiff_soilgrid \
  --runner DirectRunner \
  --project <your project> \
  --temp_location <your temp bucket> \
  --gcs_url gs://geobeam/examples/soilgrid-test-clipped.tif \
  --dataset examples \
  --table soilgrid \
  --band_column h3
```

4.4.2 Run in Dataflow

```
python -m geobeam.examples.geotiff_soilgrid \
  --runner DataflowRunner \
  --worker_harness_container_image gcr.io/dataflow-geobeam/example \
  --experiment use_runner_v2 \
  --temp_location <your temp bucket> \
  --project <your project> \
  --service_account_email <your service account> \
  --region us-central1 \
  --machine_type c2-standard-8 \
  --gcs_url gs://geobeam/examples/soilgrid-test-clipped.tif \
  --merge_blocks 20 \
  --dataset examples \
  --table soilgrid \
  --band_column h3
```

4.5 shapefile_nfhl

4.5.1 Run Locally

```
python -m geobeam.examples.shapefile_nfhl \
  --runner DirectRunner \
  --project <your project> \
  --temp_location <your temp bucket> \
  --gcs_url gs://geobeam/examples/510104_20170217.zip \
  --dataset examples \
  --table FLD_HAZ_AR \
  --layer_name S_FLD_HAZ_AR
```

4.5.2 Run in Dataflow

```
python -m geobeam.examples.shapefile_nfhl \
  --runner DataflowRunner \
  --project <your project> \
  --temp_location <your temp bucket> \
  --worker_harness_container_image gcr.io/dataflow-geobeam/example \
  --experiment use_runner_v2 \
  --service_account_email <your service account> \
  --gcs_url gs://geobeam/examples/510104_20170217.zip \
  --layer_name S_FLD_HAZ_AR \
  --dataset examples \
  --table FLD_HAZ_AR
```

4.6 streaming_pubsub

4.6.1 Run Locally

4.6.1.1 setup pubsub emulator

```
gcloud components install pubsub-emulator
gcloud beta emulators pubsub start &
curl -X PUT -v http://localhost:8085/v1/projects/example-project/topics/example-topic
```

4.6.1.2 run pipeline

```
PUBSUB_EMULATOR_HOST=localhost:8085 python -m geobeam.examples.streaming_pubsub \
  --runner DirectRunner \
  --streaming \
  --in_proj "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0␣
↪+k=1.0 +units=m +nadgrids=@null +wktext  +no_defs"
```

4.6.1.3 publish messages to topic

```
curl -v \
  -X POST \
  -H "Content-Type: application/json" \
  -d "@geobeam/examples/pubsub_emulator_messages.json" \
  http://localhost:8085/v1/projects/example-project/topics/example-topic:publish
```

4.6.1.4 unfortunate note

There is a bug in the pubsub emulator, or the directrunner, that garbles the message timestamp from the emulator. You may not be able to use the emulator for testing until this is resolved.

4.7 geojson_stormwater

```
bq mk --table <dataset>.stormwater geobeam/examples/stormwater_schema.json
```

4.7.1 Run Locally

```
python -m geobeam.examples.geojson_stormwater \
  --runner DirectRunner \
  --project <your project> \
  --temp_location <your temp bucket> \
  --gcs_url gs://geobeam/examples/Stormwater_Pipes.geojson \
  --dataset examples \
  --table stormwater \
```

4.7.2 Run in Dataflow

```
python -m geobeam.examples.geojson_stormwater \
  --runner DataflowRunner \
  --project <your project> \
  --temp_location <your temp bucket> \
  --worker_harness_container_image gcr.io/dataflow-geobeam/example \
  --experiment use_runner_v2 \
  --service_account_email <your service account> \
  --gcs_url gs://geobeam/examples/Stormwater_Pipes.geojson \
  --dataset examples \
  --table stormwater
```

CHAPTER

FIVE

INDICES AND TABLES

- search

PYTHON MODULE INDEX

g

INDEX