Cloud AutoML Vision

REST Resource: projects.locations.datasets.tableSpecs

Resource: TableSpec

A specification of a relational table. The table's schema is represented via its child column specs. It is pre-populated as part of datasets.importData by schema inference algorithm, the version of which is a required parameter of datasets.importData InputConfig. Note: While working with a table, at times the schema may be inconsistent with the data in the table (e.g. string in a FLOAT64 column). The consistency validation is done upon creation of a model. Used by: * Tables

JSON representation

```
{
  "name": string,
  "timeColumnSpecId": string,
  "rowCount": string,
  "validRowCount": string,
  "columnCount": string,
  "inputConfigs": [
    {
      object (InputConfig (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/pro
    }
  ],
  "etag": string
}
```

Fields	
name	**string** Output only. The resource name of the table spec. Form: **projects/{project_id}/locations/{locationId}/datasets/{datasetId}/tableSpecs/{tableSpecId}**

Fields	
`timeColumnSpecId`	`string` columnSpecId of the time column. Only used if the parent dataset's mlUseColumnSpecId is not set. Used to split rows into TRAIN, VALIDATE and TEST sets such that oldest rows go to TRAIN set, newest to TEST, and those in between to VALIDATE. Required type: TIMESTAMP. If both this column and ml_use_column are not set, then ML use of all rows will be assigned by AutoML. NOTE: Updates of this field will instantly affect any other users concurrently working with the dataset.
`rowCount`	`string (int64` (https://developers.google.com/discovery/v1/type-format) `format)` Output only. The number of rows (i.e. examples) in the table.
`validRowCount`	`string (int64` (https://developers.google.com/discovery/v1/type-format) `format)` Output only. The number of valid rows (i.e. without values that don't match DataType-s of their columns).
`columnCount`	`string (int64` (https://developers.google.com/discovery/v1/type-format) `format)` Output only. The number of columns of the table. That is, the number of child ColumnSpec-s.
`inputConfigs[]`	`object (InputConfig` (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs#InputConfig) `)` Output only. Input configs via which data currently residing in the table had been imported.
`etag`	`string` Used to perform consistent read-modify-write updates. If not set, a blind "overwrite" update happens.

InputConfig

Input configuration for datasets.importData Action.

The format of input depends on dataset_metadata the Dataset into which the import is happening has. As input source the `gcsSource` (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs#InputConfig.FIELDS.gcs_source) is expected, unless specified otherwise. Additionally any input .CSV file by itself must be 100MB or smaller, unless specified otherwise. If an "example" file (that is, image, video etc.) with identical content (even if it had different GCS_FILE_PATH) is mentioned multiple times, then its label, bounding boxes etc. are appended. The same file should be always provided with the same ML_USE and GCS_FILE_PATH, if it is not, then these values are nondeterministically selected from the given ones.

The formats are represented in EBNF with commas being literal and with non-terminal symbols defined near the end of this comment. The formats are:

- For Image Classification: CSV file(s) with each line in format: ML_USE,GCS_FILE_PATH,LABEL,LABEL,... GCS_FILE_PATH leads to image of up to 30MB in size. Supported extensions: .JPEG, .GIF, .PNG, .WEBP, .BMP, .TIFF, .ICO For MULTICLASS classification type, at most one LABEL is allowed per image. If an image has not yet been labeled, then it should be mentioned just once with no LABEL. Some sample rows: TRAIN,gs://folder/image1.jpg,daisy TEST,gs://folder/image2.jpg,dandelion,tulip,rose UNASSIGNED,gs://folder/image3.jpg,daisy UNASSIGNED,gs://folder/image4.jpg
- For Image Object Detection: CSV file(s) with each line in format: ML_USE,GCS_FILE_PATH, (LABEL,BOUNDING_BOX | ,,,,,,) GCS_FILE_PATH leads to image of up to 30MB in size. Supported extensions: .JPEG, .GIF, .PNG. Each image is assumed to be exhaustively labeled. The minimum allowed BOUNDING_BOX edge length is 0.01, and no more than 500 BOUNDING_BOX-es per image are allowed (one BOUNDING_BOX is defined per line). If an image has not yet been labeled, then it should be mentioned just once with no LABEL and the ",,,,,," in place of the BOUNDING_BOX. For images which are known to not contain any bounding boxes, they should be labelled explictly as "NEGATIVE_IMAGE", followed by ",,,,,," in place of the BOUNDING_BOX. Sample rows: TRAIN,gs://folder/image1.png,car,0.1,0.1,,,0.3,0.3,, TRAIN,gs://folder/image1.png,bike,.7,.6,,,.8,.9,, UNASSIGNED,gs://folder/im2.png,car,0.1,0.1,0.2,0.1,0.2,0.3,0.1,0.3 TEST,gs://folder/im3.png,,,,,,,, TRAIN,gs://folder/im4.png,NEGATIVE_IMAGE,,,,,,,
- For Video Classification: CSV file(s) with each line in format: ML_USE,GCS_FILE_PATH where ML_USE VALIDATE value should not be used. The GCS_FILE_PATH should lead to

another .csv file which describes examples that have given ML_USE, using the following row format: GCS_FILE_PATH,(LABEL,TIME_SEGMENT_START,TIME_SEGMENT_END | ,,) Here GCS_FILE_PATH leads to a video of up to 50GB in size and up to 3h duration. Supported extensions: .MOV, .MPEG4, .MP4, .AVI. TIME_SEGMENT_START and TIME_SEGMENT_END must be within the length of the video, and end has to be after the start. Any segment of a video which has one or more labels on it, is considered a hard negative for all other labels. Any segment with no labels on it is considered to be unknown. If a whole video is unknown, then it shuold be mentioned just once with ",," in place of LABEL, TIME_SEGMENT_START,TIME_SEGMENT_END. Sample top level CSV file: TRAIN,gs://folder/train_videos.csv TEST,gs://folder/test_videos.csv UNASSIGNED,gs://folder/other_videos.csv Sample rows of a CSV file for a particular ML_USE: gs://folder/video1.avi,car,120,180.000021 (gs://folder/video1.avi,car,120,180.000021) gs://folder/video1.avi,bike,150,180.000021 (gs://folder/video1.avi,bike,150,180.000021) gs://folder/vid2.avi,car,0,60.5 (gs://folder/vid2.avi,car,0,60.5) gs://folder/vid3.avi,, (gs://folder/vid3.avi,,),

- For Video Object Tracking: CSV file(s) with each line in format: ML_USE,GCS_FILE_PATH where ML_USE VALIDATE value should not be used. The GCS_FILE_PATH should lead to another .csv file which describes examples that have given ML_USE, using one of the following row format: GCS_FILE_PATH,LABEL, [INSTANCE_ID],TIMESTAMP,BOUNDING_BOX or GCS_FILE_PATH,,,,,,,,, Here GCS_FILE_PATH leads to a video of up to 50GB in size and up to 3h duration. Supported extensions: .MOV, .MPEG4, .MP4, .AVI. Providing INSTANCE_IDs can help to obtain a better model. When a specific labeled entity leaves the video frame, and shows up afterwards it is not required, albeit preferable, that the same INSTANCE_ID is given to it. TIMESTAMP must be within the length of the video, the BOUNDING_BOX is assumed to be drawn on the closest video's frame to the TIMESTAMP. Any mentioned by the TIMESTAMP frame is expected to be exhaustively labeled and no more than 500 BOUNDING_BOX-es per frame are allowed. If a whole video is unknown, then it should be mentioned just once with ",,,,,,,,," in place of LABEL, [INSTANCE_ID],TIMESTAMP,BOUNDING_BOX. Sample top level CSV file: TRAIN,gs://folder/train_videos.csv TEST,gs://folder/test_videos.csv UNASSIGNED,gs://folder/other_videos.csv Seven sample rows of a CSV file for a particular ML_USE: gs://folder/video1.avi,car,1,12.10,0.8,0.8,0.9,0.8,0.9,0.9,0.8,0.9 (gs://folder/video1.avi,car,1,12.10,0.8,0.8,0.9,0.8,0.9,0.9,0.8,0.9) gs://folder/video1.avi,car,1,12.90,0.4,0.8,0.5,0.8,0.5,0.9,0.4,0.9 (gs://folder/video1.avi,car,1,12.90,0.4,0.8,0.5,0.8,0.5,0.9,0.4,0.9) gs://folder/video1.avi,car,2,12.10,.4,.2,.5,.2,.5,.3,.4,.3

(gs://folder/video1.avi,car,2,12.10,.4,.2,.5,.2,.5,.3,.4,.3)
gs://folder/video1.avi,car,2,12.90,.8,.2,,,.9,.3, (gs://folder/video1.avi,car,2,12.90,.8,.2,,,.9,.3,),
gs://folder/video1.avi,bike,,12.50,.45,.45,,,.55,.55,
(gs://folder/video1.avi,bike,,12.50,.45,.45,,,.55,.55,), gs://folder/video2.avi,car,1,0,.1,.9,,,.9,.1,
(gs://folder/video2.avi,car,1,0,.1,.9,,,.9,.1,), gs://folder/video2.avi,,,,,,,,, (gs://folder/video2.avi,,,,,,,,),

- For Text Extraction: CSV file(s) with each line in format: ML_USE,GCS_FILE_PATH GCS_FILE_PATH leads to a .JSONL (that is, JSON Lines) file which either imports text in-line or as documents. Any given .JSONL file must be 100MB or smaller. The in-line .JSONL file contains, per line, a proto that wraps a TextSnippet proto (in json representation) followed by one or more AnnotationPayload protos (called annotations), which have displayName and textExtraction detail populated. The given text is expected to be annotated exhaustively, for example, if you look for animals and text contains "dolphin" that is not labeled, then "dolphin" is assumed to not be an animal. Any given text snippet content must be 10KB or smaller, and also be UTF-8 NFC encoded (ASCII already is). The document .JSONL file contains, per line, a proto that wraps a Document proto. The Document proto must have either documentText or inputConfig set. In documentText case, the Document proto may also contain the spatial information of the document, including layout, document dimension and page number. In inputConfig case, only PDF documents are supported now, and each document may be up to 2MB large. Currently, annotations on documents cannot be specified at import. Three sample CSV rows: TRAIN,gs://folder/file1.jsonl VALIDATE,gs://folder/file2.jsonl TEST,gs://folder/file3.jsonl Sample in-line JSON Lines file for entity extraction (presented here with artificial line breaks, but the only actual line break is denoted by \n).: { "document": { "documentText": {"content": "dog cat"} "layout": [{ "textSegment": { "startOffset": 0, "endOffset": 3, }, "pageNumber": 1, "boundingPoly": { "normalizedVertices": [{"x": 0.1, "y": 0.1}, {"x": 0.1, "y": 0.3}, {"x": 0.3, "y": 0.3}, {"x": 0.3, "y": 0.1},], }, "textSegmentType": TOKEN, }, { "textSegment": { "startOffset": 4, "endOffset": 7, }, "pageNumber": 1, "boundingPoly": { "normalizedVertices": [{"x": 0.4, "y": 0.1}, {"x": 0.4, "y": 0.3}, {"x": 0.8, "y": 0.3}, {"x": 0.8, "y": 0.1},], }, "textSegmentType": TOKEN, }

```
    ],
    "documentDimensions": {
      "width": 8.27,
      "height": 11.69,
      "unit": INCH,
    }
    "pageCount": 1,
  },
  "annotations": [
```

```
        {
          "displayName": "animal",
          "textExtraction": {"textSegment": {"startOffset": 0,
          "endOffset": 3}}
        },
        {
          "displayName": "animal",
          "textExtraction": {"textSegment": {"startOffset": 4,
          "endOffset": 7}}
        }
      ],
    }\n
    {
       "textSnippet": {
         "content": "This dog is good."
       },
       "annotations": [
         {
           "displayName": "animal",
           "textExtraction": {
             "textSegment": {"startOffset": 5, "endOffset": 8}
           }
         }
       ]
    }
  Sample document JSON Lines file (presented here with artificial line
  breaks, but the only actual line break is denoted by \n).:
    {
      "document": {
        "inputConfig": {
          "gcsSource": { "inputUris": [ "gs://folder/document1.pdf" ]
          }
        }
      }
    }\n
    {
      "document": {
        "inputConfig": {
          "gcsSource": { "inputUris": [ "gs://folder/document2.pdf" ]
          }
        }
      }
    }
```

- For Text Classification: CSV file(s) with each line in format: ML_USE,(TEXT_SNIPPET | GCS_FILE_PATH),LABEL,LABEL,... TEXT_SNIPPET and GCS_FILE_PATH are distinguished by a pattern. If the column content is a valid Google Cloud Storage file path, i.e. prefixed by "gs://", it will be treated as a GCS_FILE_PATH, else if the content is enclosed within double quotes (""), it is treated as a TEXT_SNIPPET. In the GCS_FILE_PATH case, the path must lead to a .txt file with UTF-8 encoding, for example, "gs://folder/content.txt", and the content in it is extracted as a text snippet. In TEXT_SNIPPET case, the column content excluding quotes is treated as to be imported text snippet. In both cases, the text snippet/file size must be within 128kB. Maximum 100 unique labels are allowed per CSV row. Sample rows: TRAIN,"They have bad food and very rude",RudeService,BadFood TRAIN,gs://folder/content.txt,SlowService TEST,"Typically always bad service there.",RudeService VALIDATE,"Stomach ache to go.",BadFood
- For Text Sentiment: CSV file(s) with each line in format: ML_USE,(TEXT_SNIPPET | GCS_FILE_PATH),SENTIMENT TEXT_SNIPPET and GCS_FILE_PATH are distinguished by a pattern. If the column content is a valid Google Cloud Storage file path, that is, prefixed by "gs://", it is treated as a GCS_FILE_PATH, otherwise it is treated as a TEXT_SNIPPET. In the GCS_FILE_PATH case, the path must lead to a .txt file with UTF-8 encoding, for example, "gs://folder/content.txt", and the content in it is extracted as a text snippet. In TEXT_SNIPPET case, the column content itself is treated as to be imported text snippet. In both cases, the text snippet must be up to 500 characters long. Sample rows: TRAIN,"@freewrytin this is way too good for your product",2 TRAIN,"I need this product so bad",3 TEST,"Thank you for this product.",4 VALIDATE,gs://folder/content.txt,2
- For Tables: Either `gcsSource` (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs#InputConfig.FIELDS.gcs_source) or

`bigquerySource` can be used. All inputs is concatenated into a single

[primary_table][google.cloud.automl.v1beta1.TablesDatasetMetadata.primary_table_name] For gcsSource: CSV file(s), where the first row of the first file is the header, containing unique column names. If the first row of a subsequent file is the same as the header, then it is also treated as a header. All other rows contain values for the corresponding columns. Each .CSV file by itself must be 10GB or smaller, and their total size must be 100GB or smaller. First three sample rows of a CSV file: "Id","First Name","Last Name","Dob","Addresses"

"1","John","Doe","1968-01-22","
[{"status":"current","address":"123_First_Avenue","city":"Seattle","state":"WA","zip":"11111","number(

{"status":"previous","address":"456_Main_Street","city":"Portland","state":"OR","zip":"22222","numbe

"2","Jane","Doe","1980-10-16","
[{"status":"current","address":"789_Any_Avenue","city":"Albany","state":"NY","zip":"33333","numberO
{"status":"previous","address":"321_Main_Street","city":"Hoboken","state":"NJ","zip":"44444","numbe
For bigquerySource: An URI of a BigQuery table. The user data size of the BigQuery table must be 100GB or smaller. An imported table must have between 2 and 1,000 columns, inclusive, and between 1000 and 100,000,000 rows, inclusive. There are at most 5 import data running in parallel. Definitions: ML_USE = "TRAIN" | "VALIDATE" | "TEST" | "UNASSIGNED" Describes how the given example (file) should be used for model training. "UNASSIGNED" can be used when user has no preference. GCS_FILE_PATH = A path to file on Google Cloud Storage, e.g. "gs://folder/image1.png". LABEL = A display name of an object on an image, video etc., e.g. "dog". Must be up to 32 characters long and can consist only of ASCII Latin letters A-Z and a-z, underscores(_), and ASCII digits 0-9. For each label an AnnotationSpec is created which displayName becomes the label; AnnotationSpecs are given back in predictions. INSTANCE_ID = A positive integer that identifies a specific instance of a labeled entity on an example. Used e.g. to track two cars on a video while being able to tell apart which one is which. BOUNDING_BOX = VERTEX,VERTEX,VERTEX,VERTEX | VERTEX,,,VERTEX,, A rectangle parallel to the frame of the example (image, video). If 4 vertices are given they are connected by edges in the order provided, if 2 are given they are recognized as diagonally opposite vertices of the rectangle. VERTEX = COORDINATE,COORDINATE First coordinate is horizontal (x), the second is vertical (y). COORDINATE = A float in 0 to 1 range, relative to total length of image or video in given dimension. For fractions the leading non-decimal 0 can be omitted (i.e. 0.3 = .3). Point 0,0 is in top left. TIME_SEGMENT_START = TIME_OFFSET Expresses a beginning, inclusive, of a time segment within an example that has a time dimension (e.g. video). TIME_SEGMENT_END = TIME_OFFSET Expresses an end, exclusive, of a time segment within an example that has a time dimension (e.g. video). TIME_OFFSET = A number of seconds as measured from the start of an example (e.g. video). Fractions are allowed, up to a microsecond precision. "inf" is allowed, and it means the end of the example. TEXT_SNIPPET = A content of a text snippet, UTF-8 encoded, enclosed within double quotes (""). SENTIMENT = An integer between 0 and Dataset.text_sentiment_dataset_metadata.sentiment_max (inclusive). Describes the ordinal of the sentiment - higher value means a more positive sentiment. All the values are completely relative, i.e. neither 0 needs to mean a negative or neutral sentiment nor sentimentMax needs to mean a positive one - it is just required that 0 is the least positive sentiment in the data, and sentimentMax is the most positive one. The SENTIMENT shouldn't be confused with "score" or "magnitude" from the previous Natural Language Sentiment Analysis API. All SENTIMENT values between 0 and sentimentMax must be represented in the imported data. On prediction the same 0 to sentimentMax range will be used. The difference between neighboring sentiment

values needs not to be uniform, e.g. 1 and 2 may be similar whereas the difference between 2 and 3 may be huge.

Errors: If any of the provided CSV files can't be parsed or if more than certain percent of CSV rows cannot be processed then the operation fails and nothing is imported. Regardless of overall success or failure the per-row failures, up to a certain count cap, is listed in Operation.metadata.partial_failures.

JSON representation

```
{
  "params": {
    string: string,
    ...
  },
  "gcsSource": {
    object (GcsSource (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.
  }
}
```

Fields

params	**map (key: string, value: string)** Additional domain-specific parameters describing the semantic of the imported data, any string must be up to 25000 characters long. • For Tables: **schema_inference_version** - (integer) Required. The version of the algorithm that should be used for the initial inference of the schema (columns' DataTypes) of the table the data is being imported into. Allowed values: "1". An object containing a list of **"key": value** pairs. Example: **{ "name": "wrench", "mass": "1.3kg", "count": "3" }**.
gcsSource	**object (GcsSource** (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs#GcsSource) **)** The Google Cloud Storage location for the input content. In datasets.importData, the gcsSource points to a csv with structure described in the comment.

GcsSource

The Google Cloud Storage location for the input content.

JSON representation

```
{
  "inputUris": [
    string
  ]
}
```

Fields

`inputUris[]`	`string` Required. Google Cloud Storage URIs to input files, up to 2000 characters long. Accepted forms: * Full object path, e.g. gs://bucket/directory/object.csv (gs://bucket/directory/object.csv)

Methods

`get` (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs/get)	Gets a table spec.
`list` (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs/list)	Lists table specs in a dataset.
`patch` (https://cloud.google.com/vision/automl/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs/patch)	Updates a table spec.