

---

# **Feast Documentation**

**Feast Authors**

**Apr 21, 2021**



# CONTENTS

<b>1</b>	<b>Feature Store</b>	<b>1</b>
<b>2</b>	<b>Config</b>	<b>5</b>
<b>3</b>	<b>Data Source</b>	<b>7</b>
<b>4</b>	<b>Entity</b>	<b>11</b>
<b>5</b>	<b>Feature View</b>	<b>13</b>
<b>6</b>	<b>Feature Table</b>	<b>15</b>
<b>7</b>	<b>Feature</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



## FEATURE STORE

```
class feast.feature_store.FeatureStore (repo_path: Optional[str] = None, config: Optional[feast.repo_config.RepoConfig] = None)
```

Bases: `object`

A FeatureStore object is used to define, create, and retrieve features.

```
apply (objects: Union[feast.entity.Entity, feast.feature_view.FeatureView, List[Union[feast.feature_view.FeatureView, feast.entity.Entity]])
```

Register objects to metadata store and update related infrastructure.

The apply method registers one or more definitions (e.g., Entity, FeatureView) and registers or updates these objects in the Feast registry. Once the registry has been updated, the apply method will update related infrastructure (e.g., create tables in an online store) in order to reflect these new definitions. All operations are idempotent, meaning they can safely be rerun.

**Args: objects (List[Union[FeatureView, Entity]]):** A list of FeatureView or Entity objects that should be registered

### Examples

```
>>> from feast.feature_store import FeatureStore >>> from
feast import Entity, FeatureView, Feature, ValueType, FileSource >>> from datetime import timedelta >>>
>>> fs = FeatureStore() >>> customer_entity = Entity(name="customer", value_type=ValueTypes.INT64,
description="customer entity") >>> customer_feature_view = FeatureView( >>> name="customer_fv",
>>> entities=["customer"], >>> features=[Feature(name="age", dtype=ValueTypes.INT64)], >>> in-
put=FileSource(path="file.parquet", event_timestamp_column="timestamp"), >>> ttl=timedelta(days=1)
>>> ) >>> fs.apply([customer_entity, customer_feature_view])
```

```
config: feast.repo_config.RepoConfig
```

```
delete_feature_view (name: str)
```

Deletes a feature view or raises an exception if not found.

**Parameters name** – Name of feature view

```
get_entity (name: str) → feast.entity.Entity
```

Retrieves an entity.

**Parameters name** – Name of entity

**Returns** Returns either the specified entity, or raises an exception if none is found

```
get_feature_view (name: str) → feast.feature_view.FeatureView
```

Retrieves a feature view.

**Parameters name** – Name of feature view

**Returns** Returns either the specified feature view, or raises an exception if none is found

**get\_historical\_features** (*entity\_df: Union[pandas.core.frame.DataFrame, str], feature\_refs: List[str]*) → `feast.infra.offline_stores.offline_store.RetrievalJob`  
 Enrich an entity dataframe with historical feature values for either training or batch scoring.

This method joins historical feature data from one or more feature views to an entity dataframe by using a time travel join.

Each feature view is joined to the entity dataframe using all entities configured for the respective feature view. All configured entities must be available in the entity dataframe. Therefore, the entity dataframe must contain all entities found in all feature views, but the individual feature views can have different entities.

Time travel is based on the configured TTL for each feature view. A shorter TTL will limit the amount of scanning that will be done in order to find feature data for a specific entity key. Setting a short TTL may result in null values being returned.

**Parameters**

- **entity\_df** (*Union[pd.DataFrame, str]*) – An entity dataframe is a collection of rows containing all entity columns (e.g., `customer_id`, `driver_id`) on which features need to be joined, as well as a `event_timestamp` column used to ensure point-in-time correctness. Either a Pandas DataFrame can be provided or a string SQL query. The query must be of a format supported by the configured offline store (e.g., BigQuery)
- **feature\_refs** – A list of features that should be retrieved from the offline store. Feature references are of the format “`feature_view:feature`”, e.g., “`customer_fv:daily_transactions`”.

**Returns** `RetrievalJob` which can be used to materialize the results.

**Examples**

```
Retrieve historical features using a BigQuery SQL entity dataframe >>> from feast.feature_store import FeatureStore >>> fs = FeatureStore(config=RepoConfig(provider="gcp")) >>> retrieval_job = fs.get_historical_features(>>> entity_df="SELECT event_timestamp, order_id, customer_id from gcp_project.my_ds.customer_orders", >>> feature_refs=["customer:age", "customer:avg_orders_1d", "customer:avg_orders_7d"] >>>) >>> feature_data = job.to_df() >>> model.fit(feature_data) # insert your modeling framework here.
```

**get\_online\_features** (*feature\_refs: List[str], entity\_rows: List[Dict[str, Any]]*) → `feast.online_response.OnlineResponse`  
 Retrieves the latest online feature data.

Note: This method will download the full feature registry the first time it is run. If you are using a remote registry like GCS or S3 then that may take a few seconds. The registry remains cached up to a TTL duration (which can be set to infinity). If the cached registry is stale (more time than the TTL has passed), then a new registry will be downloaded synchronously by this method. This download may introduce latency to online feature retrieval. In order to avoid synchronous downloads, please call `refresh_registry()` prior to the TTL being reached. Remember it is possible to set the cache TTL to infinity (cache forever).

**Parameters**

- **feature\_refs** – List of feature references that will be returned for each entity. Each feature reference should have the following format: “`feature_table:feature`” where “`feature_table`” & “`feature`” refer to the feature and feature table names respectively. Only the feature name is required.

- **entity\_rows** – A list of dictionaries where each key-value is an entity-name, entity-value pair.

**Returns** OnlineResponse containing the feature data in records.

### Examples

```
>>> from feast import FeatureStore
>>>
>>> store = FeatureStore(repo_path="...")
>>> feature_refs = ["sales:daily_transactions"]
>>> entity_rows = [{"customer_id": 0}, {"customer_id": 1}]
>>>
>>> online_response = store.get_online_features(
>>>     feature_refs, entity_rows, project="my_project")
>>> online_response_dict = online_response.to_dict()
>>> print(online_response_dict)
{'sales:daily_transactions': [1.1,1.2], 'sales:customer_id': [0,1]}
```

**list\_entities** (*allow\_cache: bool = False*) → List[feast.entity.Entity]

Retrieve a list of entities from the registry

**Parameters** **allow\_cache** (*bool*) – Whether to allow returning entities from a cached registry

**Returns** List of entities

**list\_feature\_views** () → List[feast.feature\_view.FeatureView]

Retrieve a list of feature views from the registry

**Returns** List of feature views

**materialize** (*start\_date: datetime.datetime, end\_date: datetime.datetime, feature\_views: Optional[List[str]] = None*) → None

Materialize data from the offline store into the online store.

This method loads feature data in the specified interval from either the specified feature views, or all feature views if none are specified, into the online store where it is available for online serving.

#### Parameters

- **start\_date** (*datetime*) – Start date for time range of data to materialize into the online store
- **end\_date** (*datetime*) – End date for time range of data to materialize into the online store
- **feature\_views** (*List[str]*) – Optional list of feature view names. If selected, will only run materialization for the specified feature views.

## Examples

```
Materialize all features into the online store over the interval from 3 hours ago to 10 minutes ago. >>>
from datetime import datetime, timedelta >>> from feast.feature_store import FeatureStore >>> >>> fs =
FeatureStore(config=RepoConfig(provider="gcp")) >>> fs.materialize( >>> start_date=datetime.utcnow()
- timedelta(hours=3), end_date=datetime.utcnow() - timedelta(minutes=10) >>> )
```

**materialize\_incremental** (*end\_date: datetime.datetime, feature\_views: Optional[List[str]] = None*) → None

Materialize incremental new data from the offline store into the online store.

This method loads incremental new feature data up to the specified end time from either the specified feature views, or all feature views if none are specified, into the online store where it is available for online serving. The start time of the interval materialized is either the most recent end time of a prior materialization or (now - ttl) if no such prior materialization exists.

### Parameters

- **end\_date** (*datetime*) – End date for time range of data to materialize into the online store
- **feature\_views** (*List[str]*) – Optional list of feature view names. If selected, will only run materialization for the specified feature views.

## Examples

```
Materialize all features into the online store up to 5 minutes ago. >>> from datetime
import datetime, timedelta >>> from feast.feature_store import FeatureStore >>> >>> fs =
FeatureStore(config=RepoConfig(provider="gcp", registry="gs://my-fs/", project="my_fs_proj")) >>>
fs.materialize_incremental(end_date=datetime.utcnow() - timedelta(minutes=5))
```

## property project

### refresh\_registry ()

Fetches and caches a copy of the feature registry in memory.

Explicitly calling this method allows for direct control of the state of the registry cache. Every time this method is called the complete registry state will be retrieved from the remote registry store backend (e.g., GCS, S3), and the cache timer will be reset. If `refresh_registry()` is run before `get_online_features()` is called, then `get_online_feature()` will use the cached registry instead of retrieving (and caching) the registry itself.

Additionally, the TTL for the registry cache can be set to infinity (by setting it to 0), which means that `refresh_registry()` will become the only way to update the cached registry. If the TTL is set to a value greater than 0, then once the cache becomes stale (more time than the TTL has passed), a new cache will be downloaded synchronously, which may increase latencies if the triggering method is `get_online_features()`

**repo\_path:** `pathlib.Path`

**version ()** → `str`

Returns the version of the current Feast SDK/CLI



## CONFIG

```
class feast.repo_config.DatastoreOnlineStoreConfig(*, type: typing_extensions.Literal[datastore] = 'datastore', project_id: pydantic.types.StrictStr = None)
```

Online store config for GCP Datastore

```
project_id: Optional[pydantic.types.StrictStr]  
(optional) GCP Project Id
```

```
type: typing_extensions.Literal[datastore]  
Online store type selector
```

```
class feast.repo_config.FeastBaseModel  
Feast Pydantic Configuration Class
```

```
exception feast.repo_config.FeastConfigError(error_message, config_path)
```

```
class feast.repo_config.RegistryConfig(*, path: pydantic.types.StrictStr, cache_ttl_seconds: pydantic.types.StrictInt = 600)
```

Metadata Store Configuration. Configuration that relates to reading from and writing to the Feast registry.

```
cache_ttl_seconds: pydantic.types.StrictInt
```

The cache TTL is the amount of time registry state will be cached in memory. If this TTL is exceeded then the registry will be refreshed when any feature store method asks for access to registry state. The TTL can be set to infinity by setting TTL to 0 seconds, which means the cache will only be loaded once and will never expire. Users can manually refresh the cache by calling `feature_store.refresh_registry()`

**Type** int

```
path: pydantic.types.StrictStr  
//foo/bar
```

**Type** str

**Type** Path to metadata store. Can be a local path, or remote object storage path, e.g. gcs

```
class feast.repo_config.RepoConfig(*, registry: Union[pydantic.types.StrictStr, feast.repo_config.RegistryConfig] = 'data/registry.db', project: pydantic.types.StrictStr, provider: pydantic.types.StrictStr, online_store: Union[feast.repo_config.DatastoreOnlineStoreConfig, feast.repo_config.SqliteOnlineStoreConfig] = SqliteOnlineStoreConfig(type='sqlite', path='data/online.db'))
```

Repo config. Typically loaded from `feature_store.yaml`

```
online_store: Union[feast.repo_config.DatastoreOnlineStoreConfig, feast.repo_config.SqliteOnlineStoreConfig]  
Online store configuration (optional depending on provider)
```

**Type** `OnlineStoreConfig`

**project:** `pydantic.types.StrictStr`

Feast project id. This can be any alphanumeric string up to 16 characters. You can have multiple independent feature repositories deployed to the same cloud provider account, as long as they have different project ids.

**Type** `str`

**provider:** `pydantic.types.StrictStr`

local or gcp

**Type** `str`

**registry:** `Union[pydantic.types.StrictStr, feast.repo_config.RegistryConfig]`

//foo/bar

**Type** `str`

**Type** Path to metadata store. Can be a local path, or remote object storage path, e.g. gcs

```
class feast.repo_config.SQLiteOnlineStoreConfig(*, type: typing_extensions.Literal[sqlite] = 'sqlite', path: pydantic.types.StrictStr = 'data/online.db')
```

Online store config for local (SQLite-based) store

**path:** `pydantic.types.StrictStr`

(optional) Path to sqlite db

**type:** `typing_extensions.Literal[sqlite]`

Online store type selector

## DATA SOURCE

**class** `feast.data_source.BigQueryOptions` (*table\_ref: Optional[str], query: Optional[str]*)  
DataSource BigQuery options used to source features from BigQuery query

**classmethod** `from_proto` (*bigquery\_options\_proto: feast.core.DataSource\_pb2.BigQueryOptions*)  
Creates a BigQueryOptions from a protobuf representation of a BigQuery option

**Parameters** `bigquery_options_proto` – A protobuf representation of a DataSource

**Returns** Returns a BigQueryOptions object based on the bigquery\_options protobuf

**property** `query`  
Returns the BigQuery SQL query referenced by this source

**property** `table_ref`  
Returns the table ref of this BQ table

**to\_proto** () → `feast.core.DataSource_pb2.BigQueryOptions`  
Converts an BigQueryOptionsProto object to its protobuf representation.

**Returns** BigQueryOptionsProto protobuf

**class** `feast.data_source.BigQuerySource` (*event\_timestamp\_column: str, table\_ref: Optional[str] = None, created\_timestamp\_column: Optional[str] = "", field\_mapping: Optional[Dict[str, str]] = None, date\_partition\_column: Optional[str] = "", query: Optional[str] = None*)

**property** `bigquery_options`  
Returns the bigquery options of this data source

**get\_table\_query\_string** () → `str`  
Returns a string that can directly be used to reference this table in SQL

**to\_proto** () → `feast.core.DataSource_pb2.DataSource`  
Converts an DataSourceProto object to its protobuf representation.

**class** `feast.data_source.DataSource` (*event\_timestamp\_column: str, created\_timestamp\_column: Optional[str] = "", field\_mapping: Optional[Dict[str, str]] = None, date\_partition\_column: Optional[str] = ""*)  
DataSource that can be used source features

**property** `created_timestamp_column`  
Returns the created timestamp column of this data source

**property** `date_partition_column`  
Returns the date partition column of this data source

**property event\_timestamp\_column**  
Returns the event timestamp column of this data source

**property field\_mapping**  
Returns the field mapping of this data source

**static from\_proto** (*data\_source*)  
Convert data source config in FeatureTable spec to a DataSource class object.

**to\_proto** () → *feast.core.DataSource\_pb2.DataSource*  
Converts an DataSourceProto object to its protobuf representation.

**class** *feast.data\_source.FileOptions* (*file\_format*: *Optional[feast.data\_format.FileFormat]*,  
*file\_url*: *Optional[str]*)

DataSource File options used to source features from a file

**property file\_format**  
Returns the file format of this file

**property file\_url**  
Returns the file url of this file

**classmethod from\_proto** (*file\_options\_proto*: *feast.core.DataSource\_pb2.FileOptions*)  
Creates a FileOptions from a protobuf representation of a file option

**Parameters** *file\_options\_proto* – a protobuf representation of a datasource

**Returns** Returns a FileOptions object based on the file\_options protobuf

**to\_proto** () → *feast.core.DataSource\_pb2.FileOptions*  
Converts an FileOptionsProto object to its protobuf representation.

**Returns** FileOptionsProto protobuf

**class** *feast.data\_source.FileSource* (*event\_timestamp\_column*: *str*, *file\_url*: *Optional[str]*  
*= None*, *path*: *Optional[str]* = *None*, *file\_format*:  
*Optional[feast.data\_format.FileFormat]* = *None*,  
*created\_timestamp\_column*: *Optional[str]* = "",  
*field\_mapping*: *Optional[Dict[str, str]]* = *None*,  
*date\_partition\_column*: *Optional[str]* = "")

**property file\_options**  
Returns the file options of this data source

**property path**  
Returns the file path of this feature data source

**to\_proto** () → *feast.core.DataSource\_pb2.DataSource*  
Converts an DataSourceProto object to its protobuf representation.

**class** *feast.data\_source.KafkaOptions* (*bootstrap\_servers*: *str*, *message\_format*:  
*feast.data\_format.StreamFormat*, *topic*: *str*)

DataSource Kafka options used to source features from Kafka messages

**property bootstrap\_servers**  
Returns a comma-separated list of Kafka bootstrap servers

**classmethod from\_proto** (*kafka\_options\_proto*: *feast.core.DataSource\_pb2.KafkaOptions*)  
Creates a KafkaOptions from a protobuf representation of a kafka option

**Parameters** *kafka\_options\_proto* – A protobuf representation of a DataSource

**Returns** Returns a BigQueryOptions object based on the kafka\_options protobuf

**property message\_format**

Returns the data format that is used to encode the feature data in Kafka messages

**to\_proto** () → feast.core.DataSource\_pb2.KafkaOptions

Converts an KafkaOptionsProto object to its protobuf representation.

**Returns** KafkaOptionsProto protobuf

**property topic**

Returns the Kafka topic to collect feature data from

```
class feast.data_source.KafkaSource (event_timestamp_column: str, bootstrap_servers: str,
                                     message_format: feast.data_format.StreamFormat,
                                     topic: str, created_timestamp_column: Optional[str]
                                     = "", field_mapping: Optional[Dict[str, str]] = {},
                                     date_partition_column: Optional[str] = "")
```

**property kafka\_options**

Returns the kafka options of this data source

**to\_proto** () → feast.core.DataSource\_pb2.DataSource

Converts an DataSourceProto object to its protobuf representation.

```
class feast.data_source.KinesisOptions (record_format: feast.data_format.StreamFormat, re-
                                       gion: str, stream_name: str)
DataSource Kinesis options used to source features from Kinesis records
```

**classmethod from\_proto** (kinesis\_options\_proto: feast.core.DataSource\_pb2.KinesisOptions)

Creates a KinesisOptions from a protobuf representation of a kinesis option

**Parameters kinesis\_options\_proto** – A protobuf representation of a DataSource

**Returns** Returns a KinesisOptions object based on the kinesis\_options protobuf

**property record\_format**

Returns the data format used to encode the feature data in the Kinesis records.

**property region**

Returns the AWS region of Kinesis stream

**property stream\_name**

Returns the Kinesis stream name to obtain feature data from

**to\_proto** () → feast.core.DataSource\_pb2.KinesisOptions

Converts an KinesisOptionsProto object to its protobuf representation.

**Returns** KinesisOptionsProto protobuf

```
class feast.data_source.KinesisSource (event_timestamp_column: str, cre-
                                       ated_timestamp_column: str, record_format:
                                       feast.data_format.StreamFormat, region: str,
                                       stream_name: str, field_mapping: Optional[Dict[str,
                                       str]] = {}, date_partition_column: Optional[str] = "")
```

**property kinesis\_options**

Returns the kinesis options of this data source

**to\_proto** () → feast.core.DataSource\_pb2.DataSource

Converts an DataSourceProto object to its protobuf representation.

```
class feast.data_source.SourceType (value)
```

DataSource value type. Used to define source types in DataSource.



## ENTITY

```
class feast.entity.Entity (name: str, value_type: feast.value_type.ValueType, description: str = "",  
                        join_key: Optional[str] = None, labels: Optional[MutableMapping[str,  
                        str]] = None)
```

Represents a collection of entities and associated metadata.

**property created\_timestamp**

Returns the created\_timestamp of this entity

**property description**

Returns the description of this entity

**classmethod from\_dict** (*entity\_dict*)

Creates an entity from a dict

**Parameters** *entity\_dict* – A dict representation of an entity

**Returns** Returns a EntityV2 object based on the entity dict

**classmethod from\_proto** (*entity\_proto: feast.core.Entity\_pb2.Entity*)

Creates an entity from a protobuf representation of an entity

**Parameters** *entity\_proto* – A protobuf representation of an entity

**Returns** Returns a EntityV2 object based on the entity protobuf

**classmethod from\_yaml** (*yml: str*)

Creates an entity from a YAML string body or a file path

**Parameters** *yml* – Either a file path containing a yaml file or a YAML string

**Returns** Returns a EntityV2 object based on the YAML file

**is\_valid** ()

Validates the state of a entity locally. Raises an exception if entity is invalid.

**property join\_key**

Returns the join key of this entity

**property labels**

Returns the labels of this entity. This is the user defined metadata defined as a dictionary.

**property last\_updated\_timestamp**

Returns the last\_updated\_timestamp of this entity

**property name**

Returns the name of this entity

**to\_dict** () → Dict

Converts entity to dict

**Returns** Dictionary object representation of entity

**to\_proto** () → feast.core.Entity\_pb2.Entity

Converts an entity object to its protobuf representation

**Returns** EntityV2Proto protobuf

**to\_spec\_proto** () → feast.core.Entity\_pb2.EntitySpecV2

Converts an EntityV2 object to its protobuf representation. Used when passing EntitySpecV2 object to Feast request.

**Returns** EntitySpecV2 protobuf

**to\_yaml** ()

Converts an entity to a YAML string.

**Returns** Entity string returned in YAML format

**property\_value\_type**

Returns the type of this entity



## FEATURE VIEW

```
class feast.feature_view.FeatureView (name: str, entities: List[str], features: List[feast.feature.Feature], ttl: Optional[Union[google.protobuf.duration_pb2.Duration, datetime.timedelta]], input: Union[feast.data_source.BigQuerySource, feast.data_source.FileSource], tags: Optional[Dict[str, str]] = None, online: bool = True)
```

A FeatureView defines a logical grouping of serveable features.

**classmethod from\_proto** (*feature\_view\_proto: feast.core.FeatureView\_pb2.FeatureView*)  
Creates a feature view from a protobuf representation of a feature view

**Parameters** **feature\_view\_proto** – A protobuf representation of a feature view

**Returns** Returns a FeatureViewProto object based on the feature view protobuf

**is\_valid** ()

Validates the state of a feature view locally. Raises an exception if feature view is invalid.

**to\_proto** () → *feast.core.FeatureView\_pb2.FeatureView*

Converts an feature view object to its protobuf representation.

**Returns** FeatureViewProto protobuf



## FEATURE TABLE

```
class feast.feature_table.FeatureTable (name: str, entities: List[str], features:
    List[feast.feature.Feature], batch_source: Optional[Union[feast.data_source.BigQuerySource,
    feast.data_source.FileSource]]
    = None, stream_source: Optional[Union[feast.data_source.KafkaSource,
    feast.data_source.KinesisSource]]
    = None, max_age: Optional[google.protobuf.duration_pb2.Duration]
    = None, labels: Optional[MutableMapping[str,
    str]] = None)
```

Represents a collection of features and associated metadata.

**add\_feature** (*feature: feast.feature.Feature*)

Adds a new feature to the feature table.

**property batch\_source**

Returns the batch source of this feature table

**property created\_timestamp**

Returns the created\_timestamp of this feature table

**property entities**

Returns the entities of this feature table

**property features**

Returns the features of this feature table

**classmethod from\_dict** (*ft\_dict*)

Creates a feature table from a dict

**Parameters** *ft\_dict* – A dict representation of a feature table

**Returns** Returns a FeatureTable object based on the feature table dict

**classmethod from\_proto** (*feature\_table\_proto: feast.core.FeatureTable\_pb2.FeatureTable*)

Creates a feature table from a protobuf representation of a feature table

**Parameters** *feature\_table\_proto* – A protobuf representation of a feature table

**Returns** Returns a FeatureTableProto object based on the feature table protobuf

**classmethod from\_yaml** (*yaml: str*)

Creates a feature table from a YAML string body or a file path

**Parameters** *yaml* – Either a file path containing a yaml file or a YAML string

**Returns** Returns a FeatureTable object based on the YAML file

**is\_valid()**

Validates the state of a feature table locally. Raises an exception if feature table is invalid.

**property labels**

Returns the labels of this feature table. This is the user defined metadata defined as a dictionary.

**property last\_updated\_timestamp**

Returns the last\_updated\_timestamp of this feature table

**property max\_age**

Returns the maximum age of this feature table. This is the total maximum amount of staleness that will be allowed during feature retrieval for each specific feature that is looked up.

**property name**

Returns the name of this feature table

**property stream\_source**

Returns the stream source of this feature table

**to\_dict()** → Dict

Converts feature table to dict

**Returns** Dictionary object representation of feature table

**to\_proto()** → feast.core.FeatureTable\_pb2.FeatureTable

Converts an feature table object to its protobuf representation

**Returns** FeatureTableProto protobuf

**to\_spec\_proto()** → feast.core.FeatureTable\_pb2.FeatureTableSpec

Converts an FeatureTableProto object to its protobuf representation. Used when passing FeatureTableSpecProto object to Feast request.

**Returns** FeatureTableSpecProto protobuf

**to\_yaml()**

Converts a feature table to a YAML string.

**Returns** Feature table string returned in YAML format

## FEATURE

**class** `feast.feature.Feature` (*name: str, dtype: feast.value\_type.ValueType, labels: Optional[MutableMapping[str, str]] = None*)

Feature field type

**property** `dtype`

Getter for data type of this field

**classmethod** `from_proto` (*feature\_proto: feast.core.Feature\_pb2.FeatureSpecV2*)

**Parameters** `feature_proto` – FeatureSpecV2 protobuf object

**Returns** Feature object

**property** `labels`

Getter for labels of this field

**property** `name`

Getter for name of this field

**to\_proto** () → `feast.core.Feature_pb2.FeatureSpecV2`

Converts Feature object to its Protocol Buffer representation

**class** `feast.feature.FeatureRef` (*name: str, feature\_table: str*)

Feature Reference represents a reference to a specific feature.

**classmethod** `from_proto` (*proto: feast.serving.ServingService\_pb2.FeatureReferenceV2*)

Construct a feature reference from the given FeatureReference proto Arg:

`proto`: Protobuf FeatureReference to construct from

**Returns** FeatureRef that refers to the given feature

**classmethod** `from_str` (*feature\_ref\_str: str*)

Parse the given string feature reference into FeatureRef model String feature reference should be in the format `feature_table:feature`. Where “feature\_table” and “name” are the feature\_table name and feature name respectively. :param `feature_ref_str`: String representation of the feature reference

**Returns** FeatureRef that refers to the given feature

**to\_proto** () → `feast.serving.ServingService_pb2.FeatureReferenceV2`

Convert and return this feature table reference to protobuf. :returns: Protobuf representation of this feature table reference.



## A

`add_feature()` (*feast.feature\_table.FeatureTable* method), 15

`apply()` (*feast.feature\_store.FeatureStore* method), 1

## B

`batch_source()` (*feast.feature\_table.FeatureTable* property), 15

`bigquery_options()`  
(*feast.data\_source.BigQuerySource* property), 7

`BigQueryOptions` (*class in feast.data\_source*), 7

`BigQuerySource` (*class in feast.data\_source*), 7

`bootstrap_servers()`  
(*feast.data\_source.KafkaOptions* property), 8

## C

`cache_ttl_seconds`  
(*feast.repo\_config.RegistryConfig* attribute), 5

`config` (*feast.feature\_store.FeatureStore* attribute), 1

`created_timestamp()` (*feast.entity.Entity* property), 11

`created_timestamp()`  
(*feast.feature\_table.FeatureTable* property), 15

`created_timestamp_column()`  
(*feast.data\_source.DataSource* property), 7

## D

`DataSource` (*class in feast.data\_source*), 7

`DatastoreOnlineStoreConfig` (*class in feast.repo\_config*), 5

`date_partition_column()`  
(*feast.data\_source.DataSource* property), 7

`delete_feature_view()`  
(*feast.feature\_store.FeatureStore* method), 1

`description()` (*feast.entity.Entity* property), 11

`dtype()` (*feast.feature.Feature* property), 17

## E

`entities()` (*feast.feature\_table.FeatureTable* property), 15

`Entity` (*class in feast.entity*), 11

`event_timestamp_column()`  
(*feast.data\_source.DataSource* property), 7

## F

`feast.data_source`  
module, 7

`feast.entity`  
module, 11

`feast.feature`  
module, 17

`feast.feature_store`  
module, 1

`feast.feature_table`  
module, 15

`feast.feature_view`  
module, 13

`feast.repo_config`  
module, 5

`FeastBaseModel` (*class in feast.repo\_config*), 5

`FeastConfigError`, 5

`Feature` (*class in feast.feature*), 17

`FeatureRef` (*class in feast.feature*), 17

`features()` (*feast.feature\_table.FeatureTable* property), 15

`FeatureStore` (*class in feast.feature\_store*), 1

`FeatureTable` (*class in feast.feature\_table*), 15

`FeatureView` (*class in feast.feature\_view*), 13

`field_mapping()` (*feast.data\_source.DataSource* property), 8

`file_format()` (*feast.data\_source.FileOptions* property), 8

`file_options()` (*feast.data\_source.FileSource* property), 8

`file_url()` (*feast.data\_source.FileOptions* property), 8

`FileOptions` (*class in feast.data\_source*), 8

`FileSource` (*class in feast.data\_source*), 8

[from\\_dict\(\)](#) (*feast.entity.Entity class method*), 11  
[from\\_dict\(\)](#) (*feast.feature\_table.FeatureTable class method*), 15  
[from\\_proto\(\)](#) (*feast.data\_source.BigQueryOptions class method*), 7  
[from\\_proto\(\)](#) (*feast.data\_source.DataSource static method*), 8  
[from\\_proto\(\)](#) (*feast.data\_source.FileOptions class method*), 8  
[from\\_proto\(\)](#) (*feast.data\_source.KafkaOptions class method*), 8  
[from\\_proto\(\)](#) (*feast.data\_source.KinesisOptions class method*), 9  
[from\\_proto\(\)](#) (*feast.entity.Entity class method*), 11  
[from\\_proto\(\)](#) (*feast.feature.Feature class method*), 17  
[from\\_proto\(\)](#) (*feast.feature.FeatureRef class method*), 17  
[from\\_proto\(\)](#) (*feast.feature\_table.FeatureTable class method*), 15  
[from\\_proto\(\)](#) (*feast.feature\_view.FeatureView class method*), 13  
[from\\_str\(\)](#) (*feast.feature.FeatureRef class method*), 17  
[from\\_yaml\(\)](#) (*feast.entity.Entity class method*), 11  
[from\\_yaml\(\)](#) (*feast.feature\_table.FeatureTable class method*), 15

## G

[get\\_entity\(\)](#) (*feast.feature\_store.FeatureStore method*), 1  
[get\\_feature\\_view\(\)](#) (*feast.feature\_store.FeatureStore method*), 1  
[get\\_historical\\_features\(\)](#) (*feast.feature\_store.FeatureStore method*), 2  
[get\\_online\\_features\(\)](#) (*feast.feature\_store.FeatureStore method*), 2  
[get\\_table\\_query\\_string\(\)](#) (*feast.data\_source.BigQuerySource method*), 7

## I

[is\\_valid\(\)](#) (*feast.entity.Entity method*), 11  
[is\\_valid\(\)](#) (*feast.feature\_table.FeatureTable method*), 15  
[is\\_valid\(\)](#) (*feast.feature\_view.FeatureView method*), 13

## J

[join\\_key\(\)](#) (*feast.entity.Entity property*), 11

## K

[kafka\\_options\(\)](#) (*feast.data\_source.KafkaSource property*), 9  
[KafkaOptions](#) (*class in feast.data\_source*), 8  
[KafkaSource](#) (*class in feast.data\_source*), 9  
[kinesis\\_options\(\)](#) (*feast.data\_source.KinesisSource property*), 9  
[KinesisOptions](#) (*class in feast.data\_source*), 9  
[KinesisSource](#) (*class in feast.data\_source*), 9

## L

[labels\(\)](#) (*feast.entity.Entity property*), 11  
[labels\(\)](#) (*feast.feature.Feature property*), 17  
[labels\(\)](#) (*feast.feature\_table.FeatureTable property*), 16  
[last\\_updated\\_timestamp\(\)](#) (*feast.entity.Entity property*), 11  
[last\\_updated\\_timestamp\(\)](#) (*feast.feature\_table.FeatureTable property*), 16  
[list\\_entities\(\)](#) (*feast.feature\_store.FeatureStore method*), 3  
[list\\_feature\\_views\(\)](#) (*feast.feature\_store.FeatureStore method*), 3

## M

[materialize\(\)](#) (*feast.feature\_store.FeatureStore method*), 3  
[materialize\\_incremental\(\)](#) (*feast.feature\_store.FeatureStore method*), 4  
[max\\_age\(\)](#) (*feast.feature\_table.FeatureTable property*), 16  
[message\\_format\(\)](#) (*feast.data\_source.KafkaOptions property*), 8  
[module](#)  
     [feast.data\\_source](#), 7  
     [feast.entity](#), 11  
     [feast.feature](#), 17  
     [feast.feature\\_store](#), 1  
     [feast.feature\\_table](#), 15  
     [feast.feature\\_view](#), 13  
     [feast.repo\\_config](#), 5

## N

[name\(\)](#) (*feast.entity.Entity property*), 11  
[name\(\)](#) (*feast.feature.Feature property*), 17  
[name\(\)](#) (*feast.feature\_table.FeatureTable property*), 16

## O

[online\\_store](#) (*feast.repo\_config.RepoConfig attribute*), 5



## P

path (*feast.repo\_config.RegistryConfig* attribute), 5  
 path (*feast.repo\_config.SQLiteOnlineStoreConfig* attribute), 6  
 path () (*feast.data\_source.FileSource* property), 8  
 project (*feast.repo\_config.RepoConfig* attribute), 6  
 project () (*feast.feature\_store.FeatureStore* property), 4  
 project\_id (*feast.repo\_config.DatastoreOnlineStoreConfig* attribute), 5  
 provider (*feast.repo\_config.RepoConfig* attribute), 6

## Q

query () (*feast.data\_source.BigQueryOptions* property), 7

## R

record\_format () (*feast.data\_source.KinesisOptions* property), 9  
 refresh\_registry () (*feast.feature\_store.FeatureStore* method), 4  
 region () (*feast.data\_source.KinesisOptions* property), 9  
 registry (*feast.repo\_config.RepoConfig* attribute), 6  
 RegistryConfig (class in *feast.repo\_config*), 5  
 repo\_path (*feast.feature\_store.FeatureStore* attribute), 4  
 RepoConfig (class in *feast.repo\_config*), 5

## S

SourceType (class in *feast.data\_source*), 9  
 SQLiteOnlineStoreConfig (class in *feast.repo\_config*), 6  
 stream\_name () (*feast.data\_source.KinesisOptions* property), 9  
 stream\_source () (*feast.feature\_table.FeatureTable* property), 16

## T

table\_ref () (*feast.data\_source.BigQueryOptions* property), 7  
 to\_dict () (*feast.entity.Entity* method), 11  
 to\_dict () (*feast.feature\_table.FeatureTable* method), 16  
 to\_proto () (*feast.data\_source.BigQueryOptions* method), 7  
 to\_proto () (*feast.data\_source.BigQuerySource* method), 7  
 to\_proto () (*feast.data\_source.DataSource* method), 8  
 to\_proto () (*feast.data\_source.FileOptions* method), 8  
 to\_proto () (*feast.data\_source.FileSource* method), 8

to\_proto () (*feast.data\_source.KafkaOptions* method), 9  
 to\_proto () (*feast.data\_source.KafkaSource* method), 9  
 to\_proto () (*feast.data\_source.KinesisOptions* method), 9  
 to\_proto () (*feast.data\_source.KinesisSource* method), 9  
 to\_proto () (*feast.entity.Entity* method), 12  
 to\_proto () (*feast.feature.Feature* method), 17  
 to\_proto () (*feast.feature.FeatureRef* method), 17  
 to\_proto () (*feast.feature\_table.FeatureTable* method), 16  
 to\_proto () (*feast.feature\_view.FeatureView* method), 13  
 to\_spec\_proto () (*feast.entity.Entity* method), 12  
 to\_spec\_proto () (*feast.feature\_table.FeatureTable* method), 16  
 to\_yaml () (*feast.entity.Entity* method), 12  
 to\_yaml () (*feast.feature\_table.FeatureTable* method), 16  
 topic () (*feast.data\_source.KafkaOptions* property), 9  
 type (*feast.repo\_config.DatastoreOnlineStoreConfig* attribute), 5  
 type (*feast.repo\_config.SQLiteOnlineStoreConfig* attribute), 6

## V

value\_type () (*feast.entity.Entity* property), 12  
 version () (*feast.feature\_store.FeatureStore* method), 4