Datasets

A **dataset** is the most essential representation of data from when it lands in Foundry through when it is mapped into the Ontology. Fundamentally, a dataset is a wrapper around a collection of **files** which are stored in a backing file system. The benefit of using Foundry datasets is that they provide integrated support for permission management, schema management, version control, and updates over time.

In Foundry's data integration layer, datasets are used to store and represent all types of data—structured, unstructured, and semi-structured:

- *Structured* (tabular) data consists of files that contain tabular data in an open-source format such as Parquet, along with metadata about the columns in the dataset. This metadata is stored alongside the dataset as a schema.
- *Unstructured* datasets consist of files such as images, video, or PDFs, but do not have an associated schema.
- *Semi-structured* datasets contain file formats such as XML or JSON. While it's possible to apply schemas to these file formats, it is preferable to infer a tabular schema in a downstream data transformation for performance and ease of use.

Transactions

Datasets are designed to change over time. When you open a dataset in Foundry and see rows and columns, what you are seeing is actually the latest *dataset view*.

As an end user, you will typically modify datasets using builds, which allow you to update the contents of a dataset according to logic that you specify. Behind the scenes, however, datasets are updated over time using **transactions**, which represent modifications to the files within a dataset. A transaction has a simple lifecycle:

- After a transaction is *started*, it is in an OPEN state. In this state, files can be opened and written in the dataset's backing file system.
- A transaction can be *committed*, which puts it into a COMMITTED state and any written files are now in the latest dataset view.
- A transaction can be *aborted*, which puts it into an ABORTED state. Any files that were written during the transaction are ignored.

If you are a software engineer, you may be familiar with Git. Dataset transactions are the basis of Foundry's support for data versioning, sometimes referred to as "Git for data." A transaction is analogous to a *commit* in Git: an atomic change to the contents of a dataset.

Transaction types

The way dataset files are modified in a transaction depends on the transaction type. There are four possible transaction types: SNAPSHOT, APPEND, UPDATE, and DELETE.

SNAPSHOT

A SNAPSHOT transaction replaces the current view of the dataset with a completely new set of files.

SNAPSHOT transactions are the simplest transaction type, and are the basis of batch pipelines.

APPEND

An APPEND transaction adds new files to the current dataset view.

An APPEND transaction cannot modify existing files in the current dataset view. If an APPEND transaction is opened and existing files are overwritten, then attempting to commit the transaction will fail.

APPEND transactions are the basis of incremental pipelines. By only syncing new data into Foundry and only processing this new data throughout the pipeline, changes to large datasets can be processed end-to-end in a performant way. However, building and maintaining incremental pipelines comes with additional complexity.

UPDATE

An UPDATE transaction, like an APPEND, adds new files to a dataset view, but may also overwrite the contents of existing files.

DELETE

A DELETE transaction removes files that are in the current dataset view.

Note that committing a DELETE transaction does not delete the underlying file from the backing file system—it simply removes the file reference from the dataset view.

In practice, DELETE transactions are mostly used to enable data retention workflows. By deleting files on a dataset based on a retention policy—typically based on the age of the file—data can be removed from Foundry, both to minimize storage costs and to comply with data governance requirements.

Example of transaction types

Imagine the following transaction history for a branch on a dataset, starting from the oldest:

1. SNAPSHOT contains files A and B
2. APPEND adds file C
3. UPDATE modifies file A to have different contents, A'
4. DELETE removes file B

At this point, the current dataset view would contain A' and C. If we added a fifth SNAPSHOT transaction containing file D, the current dataset view would then only

contain D (since SNAPSHOT transactions begin new views) and the first four transactions would be in an old view.

Retention

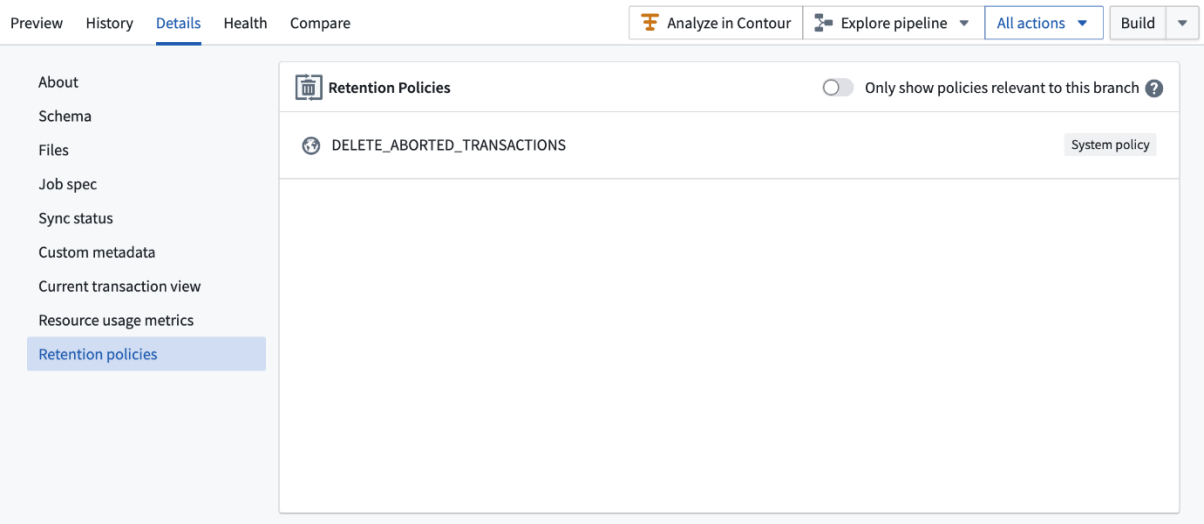
Since a DELETE transaction does not actually remove older data from the backing filesystem, you can use Retention policies to remove data in transactions which are no longer needed.

View retention policies for a dataset [Beta]

Beta

Viewing dataset retention policies is in the beta phase of development and may not be available on your enrollment. Functionality may change during active development. Contact Palantir Support to request access to dataset retention policies.

To view the retention policies that currently apply to a given dataset, navigate to the dataset details page.



Branches

While dataset transactions are designed to enable a dataset's contents to change over time, additional functionality is needed to enable *collaboration* —having multiple users experiment with changes to a dataset simultaneously. **Branches** in a dataset are designed to enable these workflows.

Dataset views

A dataset view represents the effective file contents of a dataset for a branch at a point in time. Historical views are analogous to historical versions of a dataset. To calculate which files are in a view:

1. Start with an empty set of files.

2. The view at a given time begins at the latest SNAPSHOT transaction before that point in time. If there is no SNAPSHOT transaction present, then take the earliest transaction for the dataset instead.
3. For the first transaction in the view, and for each subsequent transaction, do the following:
    - For a SNAPSHOT (which would only be the first transaction) or APPEND transaction, add all the transaction's files to the set.
    - For an UPDATE transaction, add all the transaction's files to the set and replace existing files.
    - For a DELETE transaction, remove all files in the transaction from the set.

The resulting set of files constitutes a dataset view.

If a dataset exclusively contains SNAPSHOT transactions, the number of views is equal to the number of transactions. In the case of an incremental dataset, the number of views would be equal to the number of SNAPSHOT transactions.

A view may constitute transactions from multiple branches. For example, given an incremental dataset starting with a SNAPSHOT and some APPEND transactions on the master branch, these transactions will form the start of the dataset view on the branch if subsequent transactions on the branch are also APPEND (or, strictly, *not* SNAPSHOT) transactions.

Schemas

A schema is metadata on a *dataset view* that defines how the files within the view should be interpreted. This includes how files in the view should be parsed, and how the columns or fields in the files are named or typed. The most common schemas in Foundry are *tabular*—they describe the columns in a dataset, including their names and field types.

Note that there is no guarantee that the files in a dataset actually conform to the specified schema. For example, it is possible to apply a Parquet schema to a dataset that contains CSV files. In this case, client applications attempting to read the contents of the dataset would encounter errors caused by the fact that some files do not conform to the schema.

Because schemas are stored on a *dataset view*, schemas can change over time. This is useful because the contents of a dataset may structurally change over time. For example, a new transaction may introduce a new column to a tabular dataset or change the type of a field.

In Foundry, you can view the schema on any dataset in the Dataset Preview application by navigating to the **Details** tab and selecting **Schema**.

Supported field types

Below is a list of field types available for datasets:

- BOOLEAN

- BYTE
- SHORT
- INTEGER
- LONG
- FLOAT
- DOUBLE
- DECIMAL
- STRING
- MAP
- ARRAY
- STRUCT
- BINARY
- DATE
- TIMESTAMP

Some field types require additional parameters:

- DECIMAL requires precision and scale. If you are unsure of what to set for these parameters, a good default is precision: 38 and scale: 18. 38 is the highest possible precision value.
- MAP requires mapKeyType and mapValueType, which are both field types.
- ARRAY requires arraySubType, a field type.
- STRUCT requires subSchemas, a list of field types.

Schema options

In the **Schema** section of the **Details** tab in Dataset Preview, you can add optional parsing configurations for CSV files in the options block at the bottom of the schema metadata.

File formats

Schemas include information about the underlying storage format of the files in the dataset. The three most widely-used formats are:

- Parquet
- Avro
- Text

The Text file format can be used to represent a wide range of file types, including a variety of CSV formats or JSON files. Additional information about how Text should be parsed is stored in a schema field called customMetadata.

In practice, for non-tabular formats such as JSON or XML, we recommend storing files in an unstructured (schema-less) dataset and applying a schema in a downstream data transformation.

Backing filesystem

The files tracked within a dataset are not stored in Foundry itself. Instead, a mapping is maintained between a file's *logical path* in Foundry and its *physical path* in a backing file system. The backing filesystem for Foundry is specified by a base directory in a Hadoop FileSystem. This can be a self-hosted HDFS cluster but is more commonly configured using a cloud storage provider such as Amazon S3. All dataset files are stored in a folder hierarchy below the backing file system's base directory.