+++ title = "Data frames" +++

Data frames

Grafana supports a variety of different data sources, each with its own data model. To make this possible, Grafana consolidates the query results from each of these data sources into one unified data structure called a *data frame*.

The data frame structure is a concept that's borrowed from data analysis tools like the R programming language, and Pandas.

> *Data frames are available in Grafana 7.0+, and replaced the Time series and Table structures with a more generic data structure that can support a wider range of data types.*

This document gives an overview of the data frame structure, and of how data is handled within Grafana.

The data frame

A data frame is a columnar-oriented table structure, which means it stores data by column and not by row. To understand what this means, let's look at the TypeScript definition used by Grafana:

```
interface DataFrame {
    name?:  string;
    // reference to query that create the frame
    refId?: string;

    fields: []Field;
}
```

In essence, a data frame is a collection of *fields*, where each field corresponds to a column. Each field, in turn, consists of a collection of values, along with meta information, such as the data type of those values.

```
interface Field {
  name: string;
  // Prometheus like Labels / Tags
  labels?: Record<string, string>;

  // For example string, number, time (or more specific primitives in the backend)
  type: FieldType;
  // Array of values all of the same type
  values: Vector<T>;

  // Optional display data for the field (e.g. unit, name over-ride, etc)
  config: FieldConfig;
}
```

Let's look an example. The table below demonstrates a data frame with two fields, *time* and *temperature*.

time	temperature
2020-01-02 03:04:00	45.0

2020-01-02 03:05:00	47.0
2020-01-02 03:06:00	48.0

Each field has three values, and each value in a field must share the same type. In this case, all values in the time field are timestamps, and all values in the temperature field are numbers.

One restriction on data frames is that all fields in the frame must be of the same length to be a valid data frame.

Field configuration

Each field in a data frame contains optional information about the values in the field, such as units, scaling, and so on.

By adding field configurations to a data frame, Grafana can configure visualizations automatically. For example, you could configure Grafana to automatically set the unit provided by the data source.

Transformations

Along with the type information, field configs enable *data transformations* within Grafana.

A data transformation is any function that accepts a data frame as input, and returns another data frame as output. By using data frames in your plugin, you get a range of transformations for free.

Data frames as time series

A data frame with at least one time field is considered a *time series*.

For more information on time series, refer to our [Introduction to time series]({{< relref "../../basics/timeseries.md" >}}).

Wide format

When a collection of time series shares the same _time index_—the time fields in each time series are identical—they can be stored together, in a *wide* format. By reusing the time field, we can reduce the amount of data being sent to the browser.

In this example, the `cpu` usage from each host share the time index, so we can store them in the same data frame.

```
Name: Wide
Dimensions: 3 fields by 2 rows
+----------------------+-----------------+-----------------+
| Name: time           | Name: cpu       | Name: cpu       |
| Labels:              | Labels: host=a  | Labels: host=b  |
| Type: []time.Time    | Type: []float64 | Type: []float64 |
+----------------------+-----------------+-----------------+
| 2020-01-02 03:04:00 | 3               | 4               |
| 2020-01-02 03:05:00 | 6               | 7               |
+----------------------+-----------------+-----------------+
```

However, if the two time series don't share the same time values, they are represented as two distinct data frames.

```
Name: cpu
Dimensions: 2 fields by 2 rows
+----------------------+-----------------+
| Name: time           | Name: cpu       |
| Labels:              | Labels: host=a  |
| Type: []time.Time    | Type: []float64 |
+----------------------+-----------------+
| 2020-01-02 03:04:00 | 3               |
| 2020-01-02 03:05:00 | 6               |
+----------------------+-----------------+

Name: cpu
Dimensions: 2 fields by 2 rows
+----------------------+-----------------+
| Name: time           | Name: cpu       |
| Labels:              | Labels: host=b  |
| Type: []time.Time    | Type: []float64 |
+----------------------+-----------------+
| 2020-01-02 03:04:01 | 4               |
| 2020-01-02 03:05:01 | 7               |
+----------------------+-----------------+
```

The wide format can typically be used when multiple time series are collected by the same process. In this case, every measurement is made at the same interval and will therefore share the same time values.

Long format

Some data sources return data in a *long* format (also called *narrow* format). This is common format returned by, for example, SQL databases.

In long format, string values are represented as separate fields rather than as labels. As a result, a data form in long form may have duplicated time values.

Grafana can detect and convert data frames in long format into wide format.

> ***Note:*** *Long format is currently only supported in the backend: Grafana Issue #22219.*

For example, the following data frame in long format:

```
Name: Long
Dimensions: 4 fields by 4 rows
+----------------------+-----------------+-----------------+----------------+
| Name: time           | Name: aMetric   | Name: bMetric   | Name: host     |
| Labels:              | Labels:         | Labels:         | Labels:        |
| Type: []time.Time    | Type: []float64 | Type: []float64 | Type: []string |
+----------------------+-----------------+-----------------+----------------+
| 2020-01-02 03:04:00 | 2               | 10              | foo            |
| 2020-01-02 03:04:00 | 5               | 15              | bar            |
| 2020-01-02 03:05:00 | 3               | 11              | foo            |
| 2020-01-02 03:05:00 | 6               | 16              | bar            |
+----------------------+-----------------+-----------------+----------------+
```

can be converted into a data frame in wide format:

```
Name: Wide
Dimensions: 5 fields by 2 rows
+---------------------+------------------+------------------+------------------+----
--------------+
| Name: time          | Name: aMetric    | Name: bMetric    | Name: aMetric    |
Name: bMetric    |
| Labels:             | Labels: host=foo | Labels: host=foo | Labels: host=bar |
Labels: host=bar |
| Type: []time.Time   | Type: []float64  | Type: []float64  | Type: []float64  |
Type: []float64  |
+---------------------+------------------+------------------+------------------+----
--------------+
| 2020-01-02 03:04:00 | 2                | 10               | 5                | 15
|
| 2020-01-02 03:05:00 | 3                | 11               | 6                | 16
|
+---------------------+------------------+------------------+------------------+----
--------------+
```

> ***Note:*** *Not all panels support the wide time series data frame format. To keep full backward compatibility we have introduced a transformation that can be used to convert from the wide to the long format. Read more about how to use it here: [Prepare time series-transformation]({{< relref "../../panels/reference-transformation-functions.md#prepare-time-series" >}}).*

Technical references

This section contains links to technical reference and implementations of data frames.

Apache Arrow

The data frame structure is inspired by, and uses the Apache Arrow Project. Javascript Data frames use Arrow Tables as the underlying structure, and the backend Go code serializes its Frames in Arrow Tables for transmission.

Javascript

The Javascript implementation of data frames is in the `/src/dataframe` folder and `/src/types/dataframe.ts` of the `@grafana/data` package.

Go

For documentation on the Go implementation of data frames, refer to the github.com/grafana/grafana-plugin-sdk-go/data package.