# Parse Time Operator

Status: draft
Author: helin
Last update: 2019-09-09

## Objective

Introduce an operator that parses a timestamp string with the given format into the Unix time representation, the number of seconds / milliseconds / microseconds / nanoseconds elapsed since January 1, 1970 UTC.

For example, “2019-05-17T23:56:09.05Z” with the format “%Y-%m-%dT%H:%M:%E*S%Ez” gets parsed into 1558137369, when the output unit is set to SECOND.

## Motivation

Unix timestamp is widely used in ML models, especially time-series models. However, the raw timestamp is often represented in string formats (found in logs, database tables, etc) such as “2019-05-17T23:56:09.05Z”. Currently, a preprocessing step (example) outside of the TF graph is required to convert the timestamp string to the unix timestamp.

## User Benefit

Being able to parse timestamp string inside the TF graph helps moving all the preprocessing logic into the TF graph. Having all the preprocessing logic in the TF graph has the following benefits:

### Simplify training and serving

No preprocessing step is required. For example, the exported saved model can be used with TF serving for serving directly, no need to preprocess the raw data before sending prediction request to TF serving.

### Avoid potential training serving skew

Preprocessing code for training and serving might not be a shared single piece of code. For example, training preprocessing could be written in Python for ease of use, and inference preprocessing code could be written C++ for performance. Duplicate code might introduce subtle behavior differences, causing training serving skew.

### Improve the backward compatibility story

If the model's input expectation changes, the custom preprocessing binary used for serving needs to handle both old and new expectations, in order to be backward compatible with the old models.
For example, without the parse time operator, when users decide to train the model with unix timestamp in milliseconds rather than seconds. The custom code that converts timestamp string to unix timestamp needs to be handle both cases, to support both old and new models. On the contrary, with the parse time operator, converting to milliseconds or seconds is baked as an attribute of the operator in the TF graph. There is no custom binary required, so no need to worry about backward compatibility.

## Design Proposal

The Abseil library that TensorFlow depends on has a parse time function:

```
bool ParseTime(const std::string& format, const std::string& input, Time* time,
std::string* err);
```

The parse time operator will mostly be a wrapper around it. The difference will be `absl::ParseTime` outputs `absl::Time`, and the parse time operator outputs `int64` timestamp with the specified unit in second, millisecond, microsecond or nanosecond:

```
REGISTER_OP("ParseTime")
.Input("time_string: string")
    .Output("time_int64: int64")
    .Attr("time_format: string")
    .Attr("output_unit: {'SECOND', 'MILLISECOND', 'MICROSECOND', 'NANOSECOND'}")
    .SetShapeFn(tensorflow::shape_inference::UnchangedShape)
```

Internally, the parse time operator converts `absl::Time` to `int64` using `absl::ToUnixSeconds`, `absl::ToUnixMillis`, etc:

```
switch (output_unit_) {
  case SECOND:
    output_flat(i) = absl::ToUnixSeconds(time);
    break;
  case MILLISECOND:
    output_flat(i) = absl::ToUnixMillis(time);
    break;
  case MICROSECOND:
    output_flat(i) = absl::ToUnixMicros(time);
```

```
    break;
  case NANOSECOND:
    output_flat(i) = absl::ToUnixNanos(time);
    break;
}
```