tf.parse_example

```
parse_example(
    serialized,
    features,
    name=None,
    example_names=None
)
```

Defined in `tensorflow/python/ops/parsing_ops.py`.

See the guides: Inputs and Readers > Converting, Reading data > Reading from files

Parses `Example` protos into a `dict` of tensors.

Parses a number of serialized `Example` protos given in `serialized`. We refer to `serialized` as a batch with `batch_size` many entries of individual `Example` protos.

`example_names` may contain descriptive names for the corresponding serialized protos. These may be useful for debugging purposes, but they have no effect on the output. If not `None`, `example_names` must be the same length as `serialized`.

This op parses serialized examples into a dictionary mapping keys to `Tensor` and `SparseTensor` objects. `features` is a dict from keys to `VarLenFeature`, `SparseFeature`, and `FixedLenFeature` objects. Each `VarLenFeature` and `SparseFeature` is mapped to a `SparseTensor`, and each `FixedLenFeature` is mapped to a `Tensor`.

Each `VarLenFeature` maps to a `SparseTensor` of the specified type representing a ragged matrix. Its indices are `[batch, index]` where `batch` identifies the example in `serialized`, and `index` is the value's index in the list of values associated with that feature and example.

Each `SparseFeature` maps to a `SparseTensor` of the specified type representing a Tensor of `dense_shape` `[batch_size] + SparseFeature.size`. Its `values` come from the feature in the examples with key `value_key`. A `values[i]` comes from a position `k` in the feature of an example at batch entry `batch`. This positional information is recorded in `indices[i]` as `[batch, index_0, index_1, ...]` where `index_j` is the `k-th` value of the feature in the example at with key `SparseFeature.index_key[j]. In other words, we split the indices (except the first index indicating the batch entry) of a` SparseTensor `by dimension into different features of the` Example `. Due to its complexity a` VarLenFeature `should be preferred over a` SparseFeature` whenever possible.

Each `FixedLenFeature` `df` maps to a `Tensor` of the specified type (or `tf.float32` if not specified) and shape `(serialized.size(),) + df.shape`.

`FixedLenFeature` entries with a `default_value` are optional. With no default value, we will fail if that `Feature` is missing from any example in `serialized`.

Each `FixedLenSequenceFeature` `df` maps to a `Tensor` of the specified type (or `tf.float32` if not specified) and shape `(serialized.size(), None) + df.shape`. All examples in `serialized` will be padded with `default_value` along the second dimension.

Examples:

For example, if one expects a `tf.float32` `VarLenFeature` `ft` and three serialized `Example`s are provided:

```
serialized = [
  features
    { feature { key: "ft" value { float_list { value: [1.0, 2.0] } } } },
  features
    { feature []},
  features
    { feature { key: "ft" value { float_list { value: [3.0] } } }
]
```

then the output will look like:

```
{"ft": SparseTensor(indices=[[0, 0], [0, 1], [2, 0]],
                    values=[1.0, 2.0, 3.0],
                    dense_shape=(3, 2)) }
```

If instead a `FixedLenSequenceFeature` with `default_value = -1.0` and `shape=[]` is used then the output will look like:

```
{"ft": [[1.0, 2.0], [3.0, -1.0]]}
```

Given two `Example` input protos in `serialized`:

```
[
  features {
    feature { key: "kw" value { bytes_list { value: [ "knit", "big" ] } } }
    feature { key: "gps" value { float_list { value: [] } } }
  },
  features {
    feature { key: "kw" value { bytes_list { value: [ "emmy" ] } } }
    feature { key: "dank" value { int64_list { value: [ 42 ] } } }
    feature { key: "gps" value { } }
  }
]
```

And arguments

```
example_names: ["input0", "input1"],
features: {
    "kw": VarLenFeature(tf.string),
    "dank": VarLenFeature(tf.int64),
    "gps": VarLenFeature(tf.float32),
}
```

Then the output is a dictionary:

```
{
  "kw": SparseTensor(
      indices=[[0, 0], [0, 1], [1, 0]],
      values=["knit", "big", "emmy"]
      dense_shape=[2, 2]),
  "dank": SparseTensor(
      indices=[[1, 0]],
      values=[42],
      dense_shape=[2, 1]),
  "gps": SparseTensor(
      indices=[],
      values=[],
      dense_shape=[2, 0]),
}
```

For dense results in two serialized `Example`s:

```
[
  features {
    feature { key: "age" value { int64_list { value: [ 0 ] } } }
    feature { key: "gender" value { bytes_list { value: [ "f" ] } } }
   },
   features {
    feature { key: "age" value { int64_list { value: [] } } }
    feature { key: "gender" value { bytes_list { value: [ "f" ] } } }
  }
]
```

We can use arguments:

```
example_names: ["input0", "input1"],
features: {
    "age": FixedLenFeature([], dtype=tf.int64, default_value=-1),
    "gender": FixedLenFeature([], dtype=tf.string),
}
```

And the expected output is:

```
{
  "age": [[0], [-1]],
  "gender": [["f"], ["f"]],
}
```

An alternative to `VarLenFeature` to obtain a `SparseTensor` is `SparseFeature`. For example, given two `Example` input protos in `serialized`:

```
[
  features {
    feature { key: "val" value { float_list { value: [ 0.5, -1.0 ] } } }
    feature { key: "ix" value { int64_list { value: [ 3, 20 ] } } }
  },
  features {
    feature { key: "val" value { float_list { value: [ 0.0 ] } } }
    feature { key: "ix" value { int64_list { value: [ 42 ] } } }
  }
]
```

And arguments

```
example_names: ["input0", "input1"],
features: {
    "sparse": SparseFeature(
        index_key="ix", value_key="val", dtype=tf.float32, size=100),
}
```

Then the output is a dictionary:

```
{
  "sparse": SparseTensor(
      indices=[[0, 3], [0, 20], [1, 42]],
      values=[0.5, -1.0, 0.0]
      dense_shape=[2, 100]),
}
```

Args:

- `serialized`: A vector (1-D Tensor) of strings, a batch of binary serialized `Example` protos.

- `features`: A `dict` mapping feature keys to `FixedLenFeature`, `VarLenFeature`, and `SparseFeature` values.
- `name`: A name for this operation (optional).
- `example_names`: A vector (1-D Tensor) of strings (optional), the names of the serialized protos in the batch.

Returns:

A `dict` mapping feature keys to `Tensor` and `SparseTensor` values.

Raises:

- `ValueError`: if any feature is invalid.

Last updated November 2, 2017.