

## tf.RandomShuffleQueue

## Contents

Class RandomShuffleQueue

Properties

dtypes

name

Class **RandomShuffleQueue**Inherits From: [QueueBase](#)Defined in [tensorflow/python/ops/data\\_flow\\_ops.py](#).

See the guides: [Inputs and Readers > Queues](#), [Reading data > Reading from files](#), [Threading and Queues > Queue usage overview](#)

A queue implementation that dequeues elements in a random order.

See [tf.QueueBase](#) for a description of the methods on this class.

## Properties

**dtypes**

The list of dtypes for each component of a queue element.

**name**

The name of the underlying queue.

**names**

The list of names for each component of a queue element.

**queue\_ref**

The underlying queue reference.

**shapes**

The list of shapes for each component of a queue element.

## Methods

## `__init__`

```
__init__(
    capacity,
    min_after_dequeue,
    dtypes,
    shapes=None,
    names=None,
    seed=None,
    shared_name=None,
    name='random_shuffle_queue'
)
```

Create a queue that dequeues elements in a random order.

A `RandomShuffleQueue` has bounded capacity; supports multiple concurrent producers and consumers; and provides exactly-once delivery.

A `RandomShuffleQueue` holds a list of up to `capacity` elements. Each element is a fixed-length tuple of tensors whose dtypes are described by `dtypes`, and whose shapes are optionally described by the `shapes` argument.

If the `shapes` argument is specified, each component of a queue element must have the respective fixed shape. If it is unspecified, different queue elements may have different shapes, but the use of `dequeue_many` is disallowed.

The `min_after_dequeue` argument allows the caller to specify a minimum number of elements that will remain in the queue after a `dequeue` or `dequeue_many` operation completes, to ensure a minimum level of mixing of elements. This invariant is maintained by blocking those operations until sufficient elements have been enqueued. The `min_after_dequeue` argument is ignored after the queue has been closed.

### Args:

- `capacity`: An integer. The upper bound on the number of elements that may be stored in this queue.
- `min_after_dequeue`: An integer (described above).
- `dtypes`: A list of `DType` objects. The length of `dtypes` must equal the number of tensors in each queue element.
- `shapes`: (Optional.) A list of fully-defined `TensorShape` objects with the same length as `dtypes`, or `None`.
- `names`: (Optional.) A list of string naming the components in the queue with the same length as `dtypes`, or `None`. If specified the dequeue methods return a dictionary with the names as keys.
- `seed`: A Python integer. Used to create a random seed. See `tf.set_random_seed` for behavior.
- `shared_name`: (Optional.) If non-empty, this queue will be shared under the given name across multiple sessions.
- `name`: Optional name for the queue operation.

## `close`

```
close(
    cancel_pending_enqueues=False,
    name=None
)
```

Closes this queue.

This operation signals that no more elements will be enqueued in the given queue. Subsequent `enqueue` and `enqueue_many` operations will fail. Subsequent `dequeue` and `dequeue_many` operations will continue to succeed if sufficient elements remain in the queue. Subsequently dequeue and dequeue\_many operations that would otherwise block waiting for more elements (if close hadn't been called) will now fail immediately.

If `cancel_pending_enqueues` is `True`, all pending requests will also be canceled.

Args:

- `cancel_pending_enqueues`: (Optional.) A boolean, defaulting to `False` (described above).
- `name`: A name for the operation (optional).

Returns:

The operation that closes the queue.

## dequeue

```
dequeue(name=None)
```

Dequeues one element from this queue.

If the queue is empty when this operation executes, it will block until there is an element to dequeue.

At runtime, this operation may raise an error if the queue is `tf.QueueBase.close` before or during its execution. If the queue is closed, the queue is empty, and there are no pending enqueue operations that can fulfill this request, `tf.errors.OutOfRangeError` will be raised. If the session is `tf.Session.close`, `tf.errors.CancelledError` will be raised.

Args:

- `name`: A name for the operation (optional).

Returns:

The tuple of tensors that was dequeued.

## dequeue\_many

```
dequeue_many(  
    n,  
    name=None  
)
```

Dequeues and concatenates `n` elements from this queue.

This operation concatenates queue-element component tensors along the 0th dimension to make a single component tensor. All of the components in the dequeued tuple will have size `n` in the 0th dimension.

If the queue is closed and there are less than `n` elements left, then an `OutOfRange` exception is raised.

At runtime, this operation may raise an error if the queue is `tf.QueueBase.close` before or during its execution. If the queue is closed, the queue contains fewer than `n` elements, and there are no pending enqueue operations that can fulfill this request, `tf.errors.OutOfRangeError` will be raised. If the session is `tf.Session.close`, `tf.errors.CancelledError` will be raised.

Args:

- `n` : A scalar **Tensor** containing the number of elements to dequeue.
- `name` : A name for the operation (optional).

Returns:

The tuple of concatenated tensors that was dequeued.

## dequeue\_up\_to

```
dequeue_up_to(
    n,
    name=None
)
```

Dequeues and concatenates `n` elements from this queue.

**Note** This operation is not supported by all queues. If a queue does not support `DequeueUpTo`, then a **`tf.errors.UnimplementedError`** is raised.

This operation concatenates queue-element component tensors along the 0th dimension to make a single component tensor. If the queue has not been closed, all of the components in the dequeued tuple will have size `n` in the 0th dimension.

If the queue is closed and there are more than `0` but fewer than `n` elements remaining, then instead of raising a **`tf.errors.OutOfRangeError`** like **`tf.QueueBase.dequeue_many`**, less than `n` elements are returned immediately. If the queue is closed and there are `0` elements left in the queue, then a **`tf.errors.OutOfRangeError`** is raised just like in **`dequeue_many`**. Otherwise the behavior is identical to **`dequeue_many`**.

Args:

- `n` : A scalar **Tensor** containing the number of elements to dequeue.
- `name` : A name for the operation (optional).

Returns:

The tuple of concatenated tensors that was dequeued.

## enqueue

```
enqueue(
    vals,
    name=None
)
```

Enqueues one element to this queue.

If the queue is full when this operation executes, it will block until the element has been enqueued.

At runtime, this operation may raise an error if the queue is **`tf.QueueBase.close`** before or during its execution. If the queue is closed before this operation runs, **`tf.errors.CancelledError`** will be raised. If this operation is blocked, and either (i) the queue is closed by a close operation with **`cancel_pending_enqueues=True`**, or (ii) the session is **`tf.Session.close`**, **`tf.errors.CancelledError`** will be raised.

Args:

- `vals` : A tensor, a list or tuple of tensors, or a dictionary containing the values to enqueue.
- `name` : A name for the operation (optional).

Returns:

The operation that enqueues a new tuple of tensors to the queue.

## enqueue\_many

```
enqueue_many(
    vals,
    name=None
)
```

Enqueues zero or more elements to this queue.

This operation slices each component tensor along the 0th dimension to make multiple queue elements. All of the tensors in `vals` must have the same size in the 0th dimension.

If the queue is full when this operation executes, it will block until all of the elements have been enqueued.

At runtime, this operation may raise an error if the queue is `tf.QueueBase.close` before or during its execution. If the queue is closed before this operation runs, `tf.errors.CancelledError` will be raised. If this operation is blocked, and either (i) the queue is closed by a close operation with `cancel_pending_enqueues=True`, or (ii) the session is `tf.Session.close`, `tf.errors.CancelledError` will be raised.

Args:

- `vals` : A tensor, a list or tuple of tensors, or a dictionary from which the queue elements are taken.
- `name` : A name for the operation (optional).

Returns:

The operation that enqueues a batch of tuples of tensors to the queue.

## from\_list

```
from_list(
    index,
    queues
)
```

Create a queue using the queue reference from `queues[index]`.

Args:

- `index` : An integer scalar tensor that determines the input that gets selected.
- `queues` : A list of `QueueBase` objects.

Returns:

A `QueueBase` object.

Raises:

- `TypeError` : When `queues` is not a list of `QueueBase` objects, or when the data types of `queues` are not all the same.

## **is\_closed**

```
is_closed(name=None)
```

Returns true if queue is closed.

This operation returns true if the queue is closed and false if the queue is open.

Args:

- `name` : A name for the operation (optional).

Returns:

True if the queue is closed and false if the queue is open.

## **size**

```
size(name=None)
```

Compute the number of elements in this queue.

Args:

- `name` : A name for the operation (optional).

Returns:

A scalar tensor containing the number of elements in this queue.

---

*Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.*

*Last updated November 2, 2017.*

### **Stay Connected**

[Blog](#)

[GitHub](#)

[Twitter](#)

### **Support**

[Issue Tracker](#)

[Release Notes](#)

English

[Terms](#) | [Privacy](#)