

# Process Interaction — SimPy 3.0.11 documentation

## [SimPy](#)

The [Process](#) instance that is returned by [Environment.process\(\)](#) can be utilized for process interactions. The two most common examples for this are to wait for another process to finish and to interrupt another process while it is waiting for an event.

## Waiting for a Process

As it happens, a SimPy [Process](#) can be used like an **event** (technically, a process actually is an event). If you **yield** it, you are resumed once the process has finished. Imagine a car-wash simulation where cars enter the car-wash and wait for the **washing process** to finish. Or an airport simulation where passengers have to wait until a security check finishes.

Lets assume that the car from our last example magically became an electric vehicle. Electric vehicles usually take a lot of time charging their batteries after a trip. They have to wait until their battery is charged before they can start driving again.

We can model this with an additional `charge()` process for our car. Therefore, we refactor our car to be a class with two process methods: `run()` (which is the original `car()` process function) and `charge()`.

The run process is automatically started when Car is instantiated. A new charge process is started every time the vehicle starts parking. By yielding the [Process](#) instance that [Environment.process\(\)](#) returns, the run process starts waiting for it to finish:

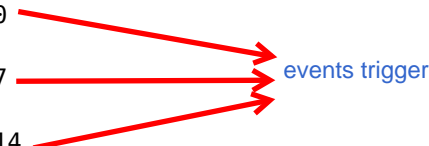
```
>>> class Car(object):
...     def __init__(self, env):
...         self.env = env
...         # Start the run process everytime an instance is created.
...         self.action = env.process(self.run())
...
...     def run(self):
...         while True:
...             print('Start parking and charging at %d' % self.env.now)
...             charge_duration = 5
...             # We yield the process that process() returns
...             # to wait for it to finish
...             yield self.env.process(self.charge(charge_duration))
...
...             # The charge process has finished and
...             # we can start driving again.
...             print('Start driving at %d' % self.env.now)
...             trip_duration = 2
```

create an event: a Process Child.  
after executing this line of code, Process Parent is suspended untill Process Child is called, Process Parant resume

```
...         yield self.env.timeout(trip_duration)
...
...     def charge(self, duration):
...         yield self.env.timeout(duration)
```

Starting the simulation is straightforward again: We create an environment, one (or more) cars and finally call [run\(\)](#).

```
>>> import simpy
>>> env = simpy.Environment()
>>> car = Car(env)
>>> env.run(until=15)
Start parking and charging at 0
Start driving at 5
Start parking and charging at 7
Start driving at 12
Start parking and charging at 14
```



## Interrupting Another Process¶

Imagine, you don't want to wait until your electric vehicle is fully charged but want to interrupt the charging process and just start driving instead.

SimPy allows you to interrupt a running process by calling its [interrupt\(\)](#) method:

```
>>> def driver(env, car):
...     yield env.timeout(3)
...     car.action.interrupt()
```

The driver process has a reference to the car's action process. After waiting for 3 time steps, it interrupts that process.

Interrupts are thrown into process functions as [Interrupt](#) exceptions that can (should) be handled by the interrupted process. The process can then decide what to do next (e.g., continuing to wait for the original event or yielding a new event):

```
>>> class Car(object):
...     def __init__(self, env):
...         self.env = env
...         self.action = env.process(self.run())
...
...     def run(self):
...         while True:
...             print('Start parking and charging at %d' % self.env.now)
...             charge_duration = 5
...             # We may get interrupted while charging the battery
...             try:
...                 yield self.env.process(self.charge(charge_duration))
...             except simpy.Interrupt:
...                 # When we received an interrupt, we stop charging and
...                 # switch to the "driving" state
...                 print('Was interrupted. Hope, the battery is full enough ...')
...
...             print('Start driving at %d' % self.env.now)
...             trip_duration = 2
...             yield self.env.timeout(trip_duration)
```

```
...
...     def charge(self, duration):
...         yield self.env.timeout(duration)
```

When you compare the output of this simulation with the previous example, you'll notice that the car now starts driving at time 3 instead of 5:

```
>>> env = simpy.Environment()
>>> car = Car(env)
>>> env.process(driver(env, car))
<Process(driver) object at 0x...>
>>> env.run(until=15)
Start parking and charging at 0
Was interrupted. Hope, the battery is full enough ...
Start driving at 3
Start parking and charging at 5
Start driving at 10
Start parking and charging at 12
```

## What's Next¶

We just demonstrated two basic methods for process interactions—waiting for a process and interrupting a process. Take a look at the [Topical Guides](#) or the [Process API](#) reference for more details.

In the [next section](#) we will cover the basic usage of shared resources.