Basic Concepts — SimPy 3.0.11 documentation

SimPy is a discrete-event simulation library. The behavior of active components (like vehicles, customers or messages) is modeled with *processes*. All processes live in an *environment*. They interact with the environment and with each other via *events*.

Processes are described by simple Python generators. You can call them *process function* or *process method*, depending on whether it is a normal function or method of a class. During their lifetime, they create events and `yield` them in order to wait for them to be triggered.

When a process yields an event, the process gets *suspended*. SimPy *resumes* the process, when the event occurs (we say that the event is *triggered*). Multiple processes can wait for the same event. SimPy resumes them in the same order in which they yielded that event.

An important event type is the `Timeout`. Events of this type are triggered after a certain amount of (simulated) time has passed. They allow a process to sleep (or hold its state) for the given time. A `Timeout` and all other events can be created by calling the appropriate method of the `Environment` that the process lives in (`Environment.timeout()` for example).

Our First Process¶

Our first example will be a *car* process. The car will alternately drive and park for a while. When it starts driving (or parking), it will print the current simulation time.

So let's start:

Environment live in, this will create Timeout event, yielded generator

```
>>> def car(env):
...     while True:
...         print('Start parking at %d' % env.now)
...         parking_duration = 5
...         yield env.timeout(parking_duration)
...
...         print('Start driving at %d' % env.now)
...         trip_duration = 2
...         yield env.timeout(trip_duration)
```

create Timeout event: parking event is timed out

create Timeout event: driving event is timed out

Our *car* process requires a reference to an `Environment` (`env`) in order to create new events. The *car*'s behavior is described in an infinite loop. Remember, this function is a generator. Though it will never terminate, it will pass the control flow back to the simulation once a `yield` statement is reached. Once the yielded event is triggered ("it occurs"), the simulation will resume the function at this statement.

As I said before, our car switches between the states *parking* and *driving*. It announces its new state by printing a message and the current simulation time (as returned by the `Environment.now` property). It then calls the `Environment.timeout()` factory function to create a `Timeout` event. This event describes the point in time the car is done *parking* (or *driving*, respectively). By yielding the event, it signals the simulation that it wants to wait for the event to occur.

Now that the behavior of our car has been modeled, lets create an instance of it and see how it behaves:

```
>>> import simpy
>>> env = simpy.Environment()
>>> env.process(car(env))
<Process(car) object at 0x...>
>>> env.run(until=15)
Start parking at 0
Start driving at 5
Start parking at 7
Start driving at 12
Start parking at 14
```

The first thing we need to do is to create an instance of `Environment`. This instance is passed into our *car* process function. Calling it creates a *process generator* that needs to be started and added to the environment via `Environment.process()`.

Note, that at this time, none of the code of our process function is being executed. Its execution is merely scheduled at the current simulation time.

The `Process` returned by `process()` can be used for process interactions (we will cover that in the next section, so we will ignore it for now).

Finally, we start the simulation by calling `run()` and passing an end time to it.

What’s Next?¶

You should now be familiar with SimPy’s terminology and basic concepts. In the next section, we will cover process interaction.