



# Manager

This lesson introduces the manager as an entity that enables seamless sharing of data and objects amongst processes.

## Manager

Python provides a way to share data between processes that may be running on different machines. The previous examples we saw of inter-process communication were restricted to a single machine. Using the **Manager** class we can share objects between processes running on the same machine or different machines.

Managers provide additional synchronization tools, such as a list or a dictionary, that can be shared between processes.

## Proxy Pattern

The manager employs the proxy pattern to enable sharing of objects across different processes. The literal definition of proxy is the authority to represent someone else. In a proxy pattern setup, a proxy is responsible for representing another object called the subject (or referent in python) in front of clients. The real subject is shielded from interacting directly with the clients.

## Base Manager



We'll start with a simple example. Imagine, we want to share a string object between two processes. The object we intend to share gets created on the machine where the manager is running. A process running on a completely different machine will be able to access the same string object. Note that the runnable script shared here is restricted to a single machine (actually a VM) but the code can be run for two processes running on two separate machines.

In our example we'll share a string object between two processes. The code below exhibits how a manager is created and shares an object:

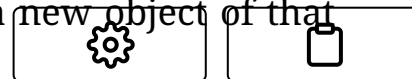
### Creating a manager and sharing a string

```
my_string = "hello World"
manager = BaseManager(address=('127.0.0.1', port_num))
manager.register('get_my_string', callable=lambda: my_string)

server = manager.get_server()
server.serve_forever()
```

- The **BaseManager** constructor takes in an **ip address** and **port number** on which it can listen for requests from proxies running on other machines. We can also leave it blank and it will default to 127.0.0.1
- In the **register()** method, the first argument passed in is called the **typeid**. The proxy object returned will have an attribute by this name. In this example the typeid is set to **get\_my\_string** and the proxy object will be able to invoke a method like **proxy\_object.get\_my\_string()**. What gets returned by the previous method invocation is specified by the next argument which is

**callable**. It can be a class name in which case a new object of that



class will be returned or it could be a method that returns an existing object. In the above example, we define a lambda function that returns the string object.

- The **server\_forever()** is a blocking call and the process running the code effectively becomes a server listening for requests for proxies. A manager object controls a server process which manages shared objects.

Now let's see how we can access the shared string in a proxy object.

## Creating a proxy and accessing a shared object

```
manager = BaseManager(address=('127.0.0.1', port_num))
manager.register('get_my_string')
manager.connect()
proxy_my_string = manager.get_my_string()

print(repr(proxy_my_string))
print(str(proxy_my_string))
```

The above code could be part of a separate python program that runs on a completely different machine.

- We create a manager object that connects to our other manager that is serving shared objects.
- Next, we need to register with the same **typeid** that we used when setting up the manager that serves shared objects, to get a proxy to the shared string object.
- Lastly, the output of the last two print statements will make it amply clear that we are indeed manipulating the shared object with a proxy.

If we run `repr(proxy_my_string)`, the proxy object will show the string

proxy. If we `repr()` the proxy object, it'll show the string



representation of the proxy object itself. However, if we `str()` the proxy object, the output will be the string representation of the referent, i.e. the shared object living on the server manager.

Below is the complete code:

## Complete Program



```
def ProcessA(port_num):
    my_string = "hello World"
    manager = BaseManager(address=('127.0.0.1', port_num))
    manager.register('get_my_string', callable=lambda: my_string)

    server = manager.get_server()
    server.serve_forever()

def ProcessB(port_num):
    manager = BaseManager(address=('127.0.0.1', port_num))
    manager.register('get_my_string')
    manager.connect()
    proxy_my_string = manager.get_my_string()

    print("In ProcessB repr(proxy_my_string) : {}".format(repr(proxy_my_string)))
    print("In ProcessB str(proxy_my_string): {}".format(str(proxy_my_string)))

    print(proxy_my_string)
    print(proxy_my_string.capitalize())
    print(proxy_my_string._callmethod("capitalize"))

if __name__ == '__main__':
    port_num = random.randint(10000, 60000)

    # Start another process which will access the shared string
    p1 = Process(target=ProcessA, args=(port_num,), name="ProcessA")
    p1.start()

    time.sleep(1)

    p2 = Process(target=ProcessB, args=(port_num,), name="ProcessB")
```

```
p2.start()
```

```
p1.join()
```

```
p2.join()
```



Run the below script and examine the output to realize the salient points we made above. Note that we have added a thread that stops the server after 3 seconds. This addition in the runnable code is only so that the code widget doesn't time out. If you copy paste the above code, it would run fine on your local machine but trying to run it in Educative's code widget would time out because the server never quits. You can ignore the additional code that stops the server for now.





```
from multiprocessing.managers import BaseManager, ListProxy
from multiprocessing import Process, Manager
from multiprocessing import current_process
from threading import Thread
import time, multiprocessing, random

def ProcessA(port_num):
    my_string = "hello World"
    manager = BaseManager(address=('127.0.0.1', port_num))
    manager.register('get_my_string', callable=lambda: my_string)
    server = manager.get_server()

    Thread(target=shutdown, args=(server,)).start()

    server.serve_forever()

def ProcessB(port_num):
    manager = BaseManager(address=('127.0.0.1', port_num))
    manager.register('get_my_string')
    manager.connect()
    proxy_my_string = manager.get_my_string()

    print("In ProcessB repr(proxy_my_string) = {}".format(repr(proxy_my_string)))
    print("In ProcessB str(proxy_my_string) = {}".format(str(proxy_my_string)))

    print(proxy_my_string)
    print(proxy_my_string.capitalize())
    print(proxy_my_string._callmethod("capitalize"))

def shutdown(server):
    time.sleep(3)
    server.stop_event.set()

if __name__ == '__main__':
    port_num = random.randint(10000, 60000)

    # Start another process which will access the shared string
    p1 = Process(target=ProcessA, args=(port_num,), name="ProcessA")
    p1.start()

    time.sleep(1)

    p2 = Process(target=ProcessB, args=(port_num,), name="ProcessB")
    p2.start()

    p1.join()
```

```
p2.join()
```



Using the proxy returned from `manager.get_my_string()` we can invoke methods on the actual object. In our example we invoke two methods using the `proxy._callmethod()` method. The server object that we get from the manager may need to be stopped at some point. Internally, the server keeps on running until an event object is set. We create a thread that sets this event object and the server shuts down.

## Shutting down server

```
def shutdown_server(server):  
    time.sleep(3)  
    server.stop_event.set()
```

It is important to realize that the `ProcessB` in our example receives only a copy of the original string defined as a local variable in the method `ProcessA`. Any changes made on the proxy object aren't reflected in `ProcessA`.

[< Back](#)

... continued


[Next >](#)


... continued



Mark as Completed



 Report an Issue

 Python & asyncio

([https://discuss.educative.io/tag/manager\\_\\_multiprocessing\\_\\_python-concurrency-for-senior-engineering-interviews](https://discuss.educative.io/tag/manager__multiprocessing__python-concurrency-for-senior-engineering-interviews))

