Application Documentation

Creating an application.

Import spacetime_local.IApplication

The application class must be derived from ***spacetime_local.IApplication.*** Each application that you create must have at least three functions written:

1. **initialize**(). This function is called when the application starts up. All object creations are pushed to the server.
2. **update**(). This function is called at every tick after a server pull objects.
3. **shutdown**(). This function is called when the application shutsdown.

Each application can declare its interaction with the pcc types. There are five main declarations that can be declared.

1. **@producer(ptype1, ptype2, …)**
    a. This declares that the application is a producer of objects of type **ptype1, ptype2**, etc, …
2. **@getter(gtype1, gtype2, …)**
    a. This declares that the application is a getter of objects of type **gtype1, gtype2**, etc, …
    b. Objects that are got, will not be tracked for changes.
3. **@setter(stype1, stype2, …)**
    a. This declares that the application is a setter of the objects of type **stype1, stype2,** etc, …
    b. Objects of this type will not be obtained each tick, unless it is declared in the **getter** or **gettersetter**. However, if these objects are dimensions of other objects that are obtained, it will be tracked for changes and updated if changed.
    c. If a join object is being used as a gettersetter, in order to change individual items in the join, those types must be declared as a **setter**.
4. **@gettersetter(gstype1, gstype2, …)**
    a. This declares that the application is a getter and a setter of the objects of type **gstype1, gstype2,** etc, …
    b. It includes the properties of both **getter** and **setter**.
    c. Objects can be obtained. Objects can be changed. Changes will be tracked.
5. **@deleter(dtype1, dtype2, …)**
    a. This declares that the application is a delete of objects of type **dtype1, dtype2,** etc, …

```
@Producer(Pedestrian)#Pedestrian is a producer
@Setter(Car,Pedestrian)#Car and Pedestrian can be set.
@GetterSetter(StoppedPedestrian, Walker, CarAndPedestrianNearby)
class PedestrianSimulation(IApplication):
  def __init__(self, frame):
    self.frame = frame
```

To start the application

You will need to import **spacetime_local.frame.frame**, and the application classes.

Instantiate an object of the frame class. Parameters that it can take in the constructor:

1. **address,** This is the address:port of the server. Defaults to **http://localhost:12000**
2. **time_step,** this is the minimum time (in milliseconds) that the frame will wait before executing the next application tick. If the time spent during processing is more than this value, it will not wait. The default value is 500ms.

Pass this frame object into the construction of each application class. The frame object that is passed through the constructor allows the application the following functions:

1. **passed_frame.get(gtype):** This function returns the list of objects of type **gtype** that it obtained at the start of the execution tick. Objects are returned only if the type is in the **getter,** or **gettersetter** declarations.
    a. Alternate form: **passed_frame.get(gtype, id):** To get one object of type **gtype** with primarykey value equal to **id.**
2. **passed_frame.add(obj):** This function adds a new object into the store at the next push cycle. Obj must be of a type that is present in the **producer** declarations.
3. **passed_frame.delete(obj):** This functions deletes obj from the store at the next push cycle. **obj** must be of a type that is present in the **delete** declarations.

Register the app to the frame by using the **your_frame.register_app(app)**. This step, registers the application with the server as well.

Start the application by called **your_frame.run() or your_frame.run_async()**. They are identical, both spawn a new thread for execution, however run will block the current thread till the execution thread has completed.

```
from spacetime_local.frame import frame
from trafficsim import TrafficSimulation
from pedestriansim_param import PedestrianSimulation
from cool_gfx import GFXSimulation
frame_car = frame(time_step = 1000) # a frame for the car is created.
frame_car.attach_app(TrafficSimulation(frame_car)) # a trafficsimulation is
created and attached
frame_ped = frame(time_step = 1000) # a frame for the pedestrian is created.
frame_ped.attach_app(PedestrianSimulation(frame_ped)) # a pedestiansimulation
is created and attached.
gfx_frame = frame(time_step = 500) # a frame for graphics is created.
gfx_frame.attach_app(GFXSimulation(gfx_frame)) # a graphics simulation app is
created and attached.
frame_car.run_async() # the car frame is executed (non blocking)
frame_ped.run_async() # the pedestrian frame is executed (non blocking)
gfx_frame.run() # the graphics frame is executed blocking
```

Things to note

1. The frame will execute initialize at every launch. This can lead to multiple objects of the same type being pushed into the server. A relaunch of the server might be in order if this was not expected behavior.