Datamodel Documentation

The datamodel in spacetime can have many classes depending on the need of the application. There are two broad categories of classes that can be included in the datamodel: ***Untracked types*** and ***Tracked types***.

Untracked Types

These are normal classes that can be created in python. These classes are not tracked directly in the spacetime server store. These classes can be used at types within other tracked types. For example, a tracked type **Car** might have a dimension **position** of type **Vector3.** Since spacetime does not have to store a collection of **Vector3** by itself**,** this class can be left as an Untracked type. There are however few constraints that the Untracked types must follow to be included as dimensions of tracked types.

1. They must be defined as a new style class in python (https://www.python.org/doc/newstyle/)
2. The class attributes must be json serializable. (**Vector3** having an attribute "x" of type **coordinate**, assuming **coordinate** is another class, is not allowed. "x" can however be an **int**)

There are no constraints on the methods, and constructors involved around these classes.

Tracked Types

These are special classes that are tracked directly, or derived dynamically in the spacetime server store. The tracked types that are currently tracked within the spacetime store include: **pcc_set, subset, join, parameterize.** Types that are going to be included are: **projection**, and **extension.**

pcc_set

This is the base type of class that is directly tracked by the spacetime server. They can be created, modified, and deleted by client applications.

Classes in the datamodel can be declared as of type pcc_set by declaring them with the decorator ***@pcc_set.***

There are several rules that must be followed for the creation of a pcc_set:

1. It must be a new style class in python.
2. Dimensions of the type have to be declared with the ***@dimension(type)*** decorator.
    a. The parameter to be passed is the datatype of the dimension.
    b. Setters of the type can be created by using the ***@<dimensionname>.setter*** decorator as shown in the example. Dimensions without a setter are considered read only dimensions.
    c. Dimensions can be defined with a default value as shown in the example.
3. One of the dimensions has be declared as a primary key with the decorator ***@primarykey(type)***
    a. A primarykey is also a dimension. The decorator ***@dimension*** need not be used if ***@primarykey*** is used.
    b. The parameter to be passed is the datatype of the primarykey.
    c. Setters of the type can be created by using the ***@<primarykeyname>.setter*** decorator as shown in the example. primarykey without a setter are considered read only primarykey.

    d. If a value is not passed during construction of an object, a unique value is generated using the uuid package in python.
4. There can be any number of methods, constructors, and other functions associated with the class. There can be additional attributes that are not tracked by the spacetime framework.
5. Each dimension can be either
    a. A primitive type.
    b. A json serializable type.
    c. A dictionary with both key and values being one of the types listed here.
    d. An object of a ***Tracked Type.***
    e. An object of an ***UnTracked Type.***
6. The class can be derived from any number of classes (multiple inheritance is supported in python)

The spacetime server will store collections of the pcc_set in its base state.

```
@pcc_set
class Car(object):
  FINAL_POSITION = 700;
  SPEED = 40;

  _ID = None
  @primarykey(str) #ID is a new primary key, primarykey acts as a property.
  def ID(self):
    return self._ID

  @ID.setter # the setter
  def ID(self, value):
    self._ID = value

  _Position = Vector3(0, 0, 0) # Vector3 is an Untracked class.
  @dimension(Vector3) # Position is a new dimension, dimension acts as a
property.
  def Position(self):
    return self._Position

  @Position.setter # the setter
  def Position(self, value):
    self._Position = value
```

subset

This is a dynamically derived type of class that is indirectly tracked by the spacetime server. They can be requested, and modified by client applications.

Classes in the datamodel can be declared as of type subset by declaring them with the decorator ***@subset(base_type).***

There are several rules that must be followed for the creation of a subset:

1. It must be a new style class in python.
2. It must be derived from the base_type.
3. Base_type has to be a ***Tracked Type*** that has been declared. (It can be any).
4. There are no dimensions that can be defined for tracking. Any dimensions created will not be tracked.

5. There are no primary keys, the primary key of the base class will be used. They shouldn't be changed.
6. There are two special functions that must be defined:
    a. **__query__**. This function has to take a list of objects all of type: **base_type** (along with the standard "self" object that python uses in class methods). It must return a list of selected objects.
    b. __predicate__. This function takes in one object of type **base_type** and returns a bool corresponding to the success of membership against the subset.
7. There can be any number of methods, and other functions associated with the class. There can be additional attributes that are not tracked by the spacetime framework.

The spacetime server will generate collections of the subset based on the query, and give it to applications that are requesting it. Changes made to it are reflected on to the corresponding object from the base class with the same primarykey.

```
@subset(Car)
class InactiveCar(Car):
  @staticmethod # has to be staticmethod
  def __query__(cars): # a function that determines the list of InactiveCars
    return [c for c in cars if InactiveCar.__predicate__(c)]

  @staticmethod
  def __predicate__(c): # the function that determines the meaning of
InactiveCar
    return c.Position == Vector3(0,0,0)

  def start(self): # method signature only for InactiveCar
    logger.debug("[InactiveCar]: {0} starting".format(self.ID))
    self.Velocity = Vector3(self.SPEED, 0, 0)
```

join

This is a dynamically derived type of class that is indirectly tracked by the spacetime server. They can be requested, and modified by client applications.

Classes in the datamodel can be declared as of type join by declaring them with the decorator ***@join(base_type1, base_type2, …).***

There are several rules that must be followed for the creation of a join:

1. It must be a new style class in python.
2. Base_types have to be a ***Tracked Type*** that has been declared. (It can be any).
3. There must be a dimension for each of the objects that must be joined together.
4. There must be a primarykey dimension
5. There are two special functions that must be defined:
    a. **__query__**. This function has to take a list of objects all of types: **base_type1, base_type2, …** (along with the standard "self" object that python uses in class methods). It must return a list of selected objects generated in the form defined like in the
    b. __predicate__. This function takes in one object of each type **base_type1, base_type2, …** and returns a bool corresponding to the success of membership against the subset.

6. There can be any number of methods, and other functions associated with the class. There can be additional attributes that are not tracked by the spacetime framework.

The spacetime server will generate collections of the join based on the query, and give it to applications that are requesting it. Changes made to it are reflected on to the corresponding object from the base class with the same primarykey.

Points of interest:

1. You can create a join of object with collection of objects, (any combination of pcc objects).

```
@join(Pedestrian, Car) # declaration
class CarAndPedestrianNearby(object): # Not inherited from anything. Choice

  @primarykey(str) # Unique primary key for this joint object.
  def ID(self):
    return self._ID

  @ID.setter
  def ID(self, value):
    self._ID = value

  @dimension(Car) # The dimension that will hold the car.
  def car(self):
    return self._car

  @car.setter
  def car(self, value):
    self._car = value

  @dimension(Pedestrian) # the dimension that will hold the pedestrian.
  def pedestrian(self):
    return self._ped

  @pedestrian.setter
  def pedestrian(self, value):
    self._ped = value

  def __init__(self, p, c): # constructor for the class, params will be put
in via Create
    self.car = c
    self.pedestrian = p

  @staticmethod
  def __query__(peds, cars): # the query, the use of Create, invokes
__init__.
    return [CarAndPedestrianNearby.Create(p, c) for p in peds for c in cars
if CarAndPedestrianNearby.__predicate__(p, c)]

  @staticmethod
  def __predicate__(p, c): # The predicate that defines the meaning of the
class.
    if abs(c.Position.X - p.X) < 130 and c.Position.Y == p.Y:
      return True
    return False
```

```
  def move(self): # method signature unique to this class.
    logger.debug("[Pedestrian]: {0} avoiding collision!".format(self.ID));
    self.pedestrian.Y += 50;
```

parameterize

This is a dynamically derived type of class that is indirectly tracked by the spacetime server. They can be requested, and modified by client applications.

Classes in the datamodel can be declared as of type parameterize by declaring them with the decorator ***@parameterize(param_type1, param_type2, …).*** Parameterize has to be declared along with either the **join** or the **subset** declarations.

There are several rules that must be followed for the creation of a parameterize:

1. It must be a new style class in python.
2. Base_types have to be a ***Tracked Type*** that has been declared. (It can be any).
3. The need for dimension, and primarykey declarations depend on the type that it is parameterizing (**join, subset**).
4. __query__, and __predicate__ takes extra parameters **param_type1_objs, param_type2_objs** etc. and can be used in any way possible.
5. There can be any number of methods, and other functions associated with the class. There can be additional attributes that are not tracked by the spacetime framework.

```
@parameterize(Car) # parameterize is sitting on top of subset.
@subset(Pedestrian) # cannot be the other way around.
class PedestrianInDanger(Pedestrian): # Rest of the declaration is almost
like subset
  @staticmethod
  def __query__(peds, cars): # cars is an extra parameter that is being
added.
    return [p for p in peds if PedestrianInDanger.__invariant__(p, cars)]

  @staticmethod
  def __predicate__(p, cars): # cars is an extra parameter that is being
added.
    for c in cars:
      if abs(c.Position.X - p.X) < 130 and c.Position.Y == p.Y:
        return True
    return False

  def move(self): # method signature just for this class.
    logger.debug("[Pedestrian]: {0} avoiding collision!".format(self.ID));
    self.Y += 50;
```

projection

This is a dynamically derived type of class that is indirectly tracked by the spacetime server. They can be created, requested, and deleted by client applications.

Classes in the datamodel can be declared as of type projection by declaring them with the decorator ***@projection(base_type).*** Only one **base_type** can be used. The methods of the base class do not follow to the projection.

There are several rules that must be followed for the creation of a projection:

1. It must be a new style class in python.
2. **base_type** has to be a ***pcc_set*** that has been declared.
3. The primarykey of the pcc_set has to be copied to the projection as shown in the example.
4. Any dimension from the pcc_set can be brought to the projection.
5. More unique dimensions can be added.
6. There are no __query__ and __predicate__ functions that are needed.

Objects of this type, when created will be placed with the pcc_set collection. When requested as a pcc_set, only dimensions that are present in the pcc_set will be exposed. When requested as a projection, only dimensions allowed in the projection are pulled. Be careful when creating new projection objects, as subsets of the pcc_set might depend on dimensions that do not exist.

There can be any number of methods, and other functions associated with the class. There can be additional attributes that are not tracked by the spacetime framework.

```
@projection(Car) #Car is a pcc_set
class CarPosition(object): # Do not derive from Car unless you want all
dimensions
  ID = Car.ID # Have to copy the primary key from Car
  Position = Car.Position # Additional dimension from Car that is needed

  @dimension(str) # New dimension just for CarPosition
  def Name(self):
    return self._name

  @Name.setter
  def Name(self, value):
    self._name = value

  def move(self): # Method signature for just CarPosition
    self.Position = Vector3(self.Position.X + 10, self.Position.Y,
self.Position.Z)

# CarPosition, and Car objects will coexist in the same set. They will typecast
depending on the call from the application.
```