OpenSRANE

Open Software for Risk Assessment of NaTech Events

by

Bijan Sayyafzadeh

As final thesis project for the degree of

Doctor of Philosophy

In

Engineering – Structure Engineering

(Developers Guideline)

Guide masters:

* Dr. Mahdi Sharifi
* Dr. Abdolreza Sarvghad Moghaddam
* Dr. Eslam Kashi

Contents

[1- Codes Description and structure 2](#_Toc105442933)

[1-1- General points 2](#_Toc105442934)

[1-2- Subpackages 2](#_Toc105442935)

[1-3- ObjManager 2](#_Toc105442936)

[1-4- \_NewClass 2](#_Toc105442937)

[1-5- \_GlobalParameters 3](#_Toc105442938)

[1-6- Modules 3](#_Toc105442939)

[1-7- How Add New Module 3](#_Toc105442940)

[2- Hazard Subpackage 3](#_Toc105442941)

[2-1- \_GlobalParameters 3](#_Toc105442942)

[1-1-1- Methods: 3](#_Toc105442943)

[2-2- Earthquake 3](#_Toc105442944)

[1-1-2- Methods: 3](#_Toc105442945)

[3- DateAndTime Subpackage 3](#_Toc105442946)

[3-1- \_GlobalParameters 3](#_Toc105442947)

[3-2- DateTime 3](#_Toc105442948)

[1-1-3- Methods: 4](#_Toc105442949)

[4- WindData Subpackage 4](#_Toc105442950)

[4-1- \_GlobalParameters 4](#_Toc105442951)

[1-1-4- Methods 4](#_Toc105442952)

[4-2- WindRose 4](#_Toc105442953)

[1-1-5- Methods 4](#_Toc105442954)

[5- Sites Subpackage 5](#_Toc105442955)

[5-1- \_GlobalParameters 5](#_Toc105442956)

[5-2- Site 5](#_Toc105442957)

[6- Substance Subpackage 5](#_Toc105442958)

[6-1- \_GlobalParameters 5](#_Toc105442959)

[6-2- Material 5](#_Toc105442960)

[6-3- DataBank 5](#_Toc105442961)

[7- Fragilities 5](#_Toc105442962)

[7-1- \_GlobalParameters 5](#_Toc105442963)

[7-2- Fragility 5](#_Toc105442964)

[7-3- Probit 5](#_Toc105442965)

[8- OutFlowModel 6](#_Toc105442966)

[8-1- Global Structure of each Subpackage 6](#_Toc105442967)

# Introduction

In this Guideline, details of the codes structure for any SubPackages are described. It is highly recommended that Developers before any develop read the target subpackage structure from this guideline to get familiar with various part of the subpackage and then start to develop.

# Code’s structure

OpenSRANE framework consist of a collection of subpackages. There two types of SubPackages, common subpackages and special subpackages. Special subpackages are the ones that are used for special purposes like Misc (That is for managing modules and special required commands like wipe), Plot and PostProcess that as are obvious from their name are for post processing activities (That maybe remove in future into another Package). Rest of the subpackages are common packages that have the following structure:

The \_\_init\_\_.py and ObjManager.py are constant modules that should exist in any subpackages and should never changes!

## \_GlobalParameters

Every subpackage contains a \_GlobalParameters.py file that involves variables and methods that is common for all classes inside the subpackage. It plays a key rule to make integrity between subpackage modules and specifies the minimum methods and variables and type of variables that each module should have.

It contains GlobalParameters class and all of the common variables and methods with no initial values and calculations are defined in this class and they show that subpackage modules should contain and fill these methods and variables.

All classes in each subpackage should inherits \_GlobalParameters class and initialize the \_GlobalParameters variables and also fill them with the results or entered parameters by the user. Also, methods that should be common for all classes in the subpackage should be define in \_GlobalParameters class. However, in any class the method code can changes by developer.

These parameters should be filled by each module and the results of calculations are stored in them. Each module can have their own internal methods and these methods are used only for internal calculations of the module and they won’t be call with other modules or subpackages. Sometimes there are some methods that maybe useful for subpackage modules so they are located in \_GlobalParameters.py and started with “\_” sign.

For some subpackages there are Calculate method in some \_GlobalParameters.py is responsible to start module calculations to fill the parameters with results. So, the main code of the module should be written in the Calculate method and this part is called by analysis subpackage and after it, the parameters should be filled by the results.

Other Common Methods: there are some other methods in \_GlobalParameters.py (That are not started with “\_”) and also these methods should be considered with each module and the need of them are explained in each subpackage description in the following chapters.

wipeAnalysis and wipeAnalysisGlobal in \_GlobalParameters.py is a very important method that shows the common parameters that should filled with their initial value at the start of the analysis.

## wipeAnalysis and wipeAnalysisGlobal

wipeAnalysisGlobal contains variables that are common between all classes and after each analysis by calling wipeAnalysis command their assigned data will be clear and the initial value will be assign to them to get ready for next analysis. These variables are those that will be use and call by the other classes in other SubPackages or call by recorders or by users to be stored for postprocess analysis.

So, by each running of the analysis these parameters will be reset and their previous values from the last analysis will be removed.

wipeAnalysis in \_GlobalParameters.py is for module parameters and it is responsible to initialize the parameters that are used only inside the module at the start of each analysis and remove the last analysis values from them. To making code briefer, the code only calls wipeAnalysis and so in each wipeAnalysis command the wipeAnalysisGlobal also should be called to reset the global parameters.

## Subpackage modules

SubPackages Modules should comply the following important rules:

* Each module should contain a class that its name is equal to the filename of the module.
* The “\_NewClass” and “\_GlobalParameters” should be imported to be inherited by the module class as shown in the following.
* The \_\_init\_\_ command should have “tag” just after self and then it is obvious that required class arguments and GlobalPrameters arguments (\_globalParametersArguments) also should be define.
* After \_\_init\_\_ definition, the 2 lines that are shown in Fix Part block in the following should be define without any changes and finally \_GlobalParameters should be initialize using the defined arguments.

**from** opensrane**.**Misc**.**\_NewClass **import** \_NewClass

**from** **.**\_GlobalParameters **import** \_GlobalParameters

**class** **moduleFilename(**\_NewClass**,**\_GlobalParameters**):**

**def** \_\_init\_\_**(**self**,**tag**,**arguments, \_globalParametersArguments**):**

#---- Fix Part for each class \_\_init\_\_ ----

ObjManager**.**Add**(**tag**,**self**)**

\_NewClass**.**\_\_init\_\_**(**self**,**tag**)**

#------------------------------------------

\_GlobalParameters**.**\_\_init\_\_**(**self, \_globalParametersArguments**)**

* After above part the class content can be written. All modules in the subpackages obey the mention rules and are proper samples to check the above rules and how they are implemented.
* Developed classes can have their own methods but they should have all methods that are defined in the \_GlobalParameters.py located in the subpackage folder and should return the parameters that are specified.

## How Add New Module

## General Rules

# Hazard Subpackage

## \_GlobalParameters

Currently no common input value has been defined for as input parameter.

### Methods:

* GetRandomMagnitude(self,rnd=None)

This common method is used for returning a magnitude value corresponding to the input random number between 0 and 1 and obviously any Hazard module should have this method for sampling. The returned value will be used in the fragilities.

## Earthquake

In this module mainly Earthquake class has been defined and, in this class, only input data get from the user and then earthquake object will be generated.

### Methods:

* GetRandomMagnitude(rnd=None):

This method is used for returning a magnitude value corresponding to the input random number between 0 and 1. Codes uses interpolate SciPy module and if user do not enter any input value program uses a uniform distribution to guess a random number between 0 and 1.

If user enter a range of magnitude values that do not fill the probability values between 0 and 1, for any random value generated less than the minimum probability value the method returns 0 as the corresponding value and for these random numbers and for random numbers greater than maximum probability value, the maximum entered magnitude will be return as the corresponding magnitude.

# DateAndTime Subpackage

## \_GlobalParameters

Have no GlobalParameters because seems there will be no additional new modules in future and any new feature can be added simply to the available module.In future if the need of the new modules feels simply the GlobalParameters will be added.

## DateTime

In this module currently Day to night ratio will be get from user by the DateTime Class.

### Methods:

* isDay():

A simple method that by generating a random number between zero and one returns a Boolean that specify that sample is in day or night.

osn.DateAndTime.ObjManager[1].isDay()

# WindData Subpackage

## \_GlobalParameters

There are some variables that should be calculated by any other modules that need to be added in this subpackage. These are as the following:

self**.**WindClass**=None**

self**.**WindDirection**=None**

self**.**WindSpeed**=None**

self**.**AlphaCOEF**=None**

self**.**isCalmn**=False**

Above parameters are obvious from users guideline and "isCalmn" is a Boolean that determine the sampled is calm condition or not. So, any module Finally should return these parameters and should fill the in the following sampling method

### Methods

* GetRandomWindِSample():

Any WindData module should have one GetRandomWindِSample() method for sampling and the result of calling this method is the returning and valuing the variables that mentioned in previous part.

## WindRose

This module contains WindRose Class and its requirements variable and methods.

### Methods

* CalcDirectionProbabilities():

This method is mostly an internal method that checks entered values for object that is generated from the WindRose class and specify calm condition ratio according user input data. This method, as said, is called internally by the other method/s.

Also, if summation of entered probability values become more than 100 this method fixes them to the100 and obviously the will be no calm condition.

* GetRandomWindِSample():

Important method that do the sampling from the user input data. At first step checks if CalcDirectionProbabilities() method didn't run, run it and then gets the sampled Day/Night. According being day or night, the related data are gotten from the WindRose object and by generating random number between zero and one wind direction and wind speed are sampled.

# Sites Subpackage

There is no special comment on this subpackage.

## \_GlobalParameters

Currently seems there is no need to any additional module for this subpackage, so all input parameters defined in the site module and there is no need to \_GlobalParameters.

## Site

This module is only for site data and there is no special algorithm to explain!

# Substance Subpackage

There is no special comment on this subpackage.

## \_GlobalParameters

Currently seems there is no need to any additional module for this subpackage, so all input parameters defined in the Material module and there is no need to \_GlobalParameters.

## Material

This module is only for site data and there is no special algorithm to explain!

## DataBank

This module is only for site data and there is no special algorithm to explain!

# Fragilities

## \_GlobalParameters

* GetProbability : The main purpose of defining fragility or probit is to get the probability of fragile, collapse, vulnerability or any type of damage under a magnitude value of external excitation. So, any fragility or probit module should have a GetProbability function that returns mentioned probability under given magnitude.

## Fragility

This module is only getting the fragility parameters from the user and calculations algorithm in its GetProbability function to return the probability under given magnitude.

## Probit

This module is only getting the probit parameters from the user and calculations algorithm in its GetProbability function to return the probability under given magnitude.

# OutFlowModel

# Connectors

* ساب پکیج­هایی که نیاز به تولید اشیاء تکراری از یک کلاس ندارند نیازی به Objmanager ندارند و بودن این فایل درون پوشه صرفاً بخاطر یکسان بودن شکل کلی پوشه­ها است. کلاس­هایی که نیاز به تولید اشیاء مشابه ندارند import از Objmanager و \_NewClass مرسوم که برای سایر کلاس­ها تعریف شده است از درون آنها حذف می­گردد و درون آنها صورت نمی­گیرد.

کد تهیه شده کاملا بصورت شی­گرایی نوشته شده است. بدان معنی که کلیه کدهای نوشته شده بصورت اشیا بوده و حتی توابع مورد نیاز نیز درون یک شی باید نوشته شود. در کنار آن نحوه کدنویسی بصورتی می­باشد که بسادگی توسط شخص سوم قابل توسعه باید باشد. به همین دلیل مقرراتی در نظر گرفته شده است که باید رعایت شوند. این نکات و قوانین به شرح زیر می­باشد:

* هر فایل تولیدی باید دارای نام مختص به خود باشد و هیچ فایل یا فلدری نباید با نام تکراری درنظر گرفته شود. مواکداً تاکید می­شود به هیچ­.جه هر کلاس یا فایل و یا پکیج جدیدی که به مجموعه می­خواهد اضافه شود، نامی تکراری نداشته باشد. پس کاربر توسعه دهنده باید در زمانی اضافه کردن فایل جدید، نامی را برای آن انتخاب کند که هیچ یک از فایل ها و فلدرها به آن نام نباشد.
* در هر فایل تنها باید یک کلاس تعریف شده باشد و نام فایل باید با نام کلاس یکسان باشد (با درنظر گرفتن حروف کوچک و بزرگ).
* نام فایل کلاس­های تعریف شده نباید با "\_" آغاز شوند و برنامه این کلاس­ها را نمی­شناسد.
* در فلدر اصلی برنامه نباید جز \_\_init\_\_.py چیز دیگری باشد و برنامه بصورت خودکار تمامی کلاس­ها را فراخوانی می­کند.
* فایل \_\_init\_\_.py نوشته شده برای شاخه اصلی برنامه با فایل­های \_\_init\_\_.py نوشته شده برای subpackageها (زیرشاخه­ها) متفاوت می­باشد.
* مکانیزم برنامه به این صورت است که کلاس­های مختلف اشیا درون فلدرها یا subpackage های جداگانه با \_\_init\_\_.py نوشته شده برای subpackageها (زیرشاخه­ها) بعد از فراخوانی برنامه در زیر همان زیرشاخه تعریف شده قابل دسترسی می­باشد اما در کنار آن کلیه کلاس­ها بصورت یکجا نیز درون روت اصلی برنامه بعد از import کردن آن قابل دسترسی می­باشد.
* در هر فلدر یا subpackage جدید باید فایل­های \_\_init\_\_ و ObjManager همانند سایر فلدرها یا subpackage های دیده شود.
* با فراخوانی برنامه کلیه کلاس­های موجود در هر زیرشاخه، درون لیستی با نام همان زیرشاخه قابل مشاهده می­باشد. همچنین درون لیستی بنام TotalClassList لیست کلیه کلاس­ها قابل دسترسی می­باشد.
* فلدر Misc شامل کلاس­هایی می­باشد که اطلاعات و ابزارهای جانبی به کاربر می­دهد و لزومی به ساخت اشیا از آن ندارد و صرفا با نام خود کلاس دسترسی به توابع موجود در آن صورت می­گیرد و شامل توابع زیر می­باشد
  + توابع موجود در کلاس MiscFuncs():
    - تابع: GetModules () برای بازگرداندن لیست ماژول­ها یا ساب پکیج­های استفاده می­گردد.
    - تابع GetClasses () : برای یازگرداندن لیست کلاس­های موجود برنامه به همراه نام ساب پکیج بالایی استفاده می­گردد.
    - تابع GetClassParrentModule(Classname): که این تابع با وارد کردن نامه کلاس مد نظر (بصورت رشته ای) نام ماژول یا سابپکیج که آن کلاس درون آن قرار دارد را باز می گرداند.
* برای هر subpackage یک کلاس ObjManager موجود می­باشد که موارد مربوط به آن به شرح زیر است:
  + وظیفه این کلاس مدیریت در تولید Tag های غیر یکسان و ذخیره نام و اشیاء تولیدی و پاک کردن کلیه اشیاء تولیدی می­باشد.
  + در شاخه اصلی برنامه که تمامی توابع و کلاس­ها قابل دسترسی می­باشد این کلاس دیده نمی­شود زیرا هر ماژول ObjManager مختص به خود را دارد.
  + در صورتی که کاربری بخواهد ماژول یا Subpackage جدیدی به برنامه اضافه کند باید این کلاس درون فلدر مربوطه افزوده گردد.
  + در ابتدای کلاس درون هر ماژول بخش­هایی ثابت تعریف شده است که باید برای کلیه کلاس­ها درنظر گرفته شود.

نکات مهم شی گرایی:

* فرض کنیم شی‑ای از یک کلاسی درست کرده ایم. مثلا از کلاس cls با ورودی های a, b شی obj رو درست کردیم:

Obj=cls(a,b)

حال اگر بخواهیم بفهمیم که شی Obj از چه کلاسی درست شده است با فرمان روبرو نام کلاس آن شی بدست می آید:

Print(obj.\_\_class\_\_.\_\_name\_\_)

که خروجی فرمان فوق همان cls خواهد بود.

اما اگر بخواهیم به خود شی کلاس cls دسترسی داشته باشیم باید از obj.\_\_class\_\_ استفاده کنیم که دقیقا معادل همان کلاس خواهد بود. بطور مثال اگر شی جدیدی را بخواهیم از کلاس مذکور درست کنیم می­توانیم بجای نام بردن نام خود کلاس از نام خود شی تولید شده قبلی استفاده کنیم:

Obj2=obj.\_\_class\_\_(a2,b2)

که بدان معنی هست که از همان کلاسی که شی obj رو درست کردی با پارامتر های a2,b2 شی جدید obj2 رو درست کن

توجه مهم که وقتم رو گرفت: برای یافتن نام کلاس یک شی که از آن کلاس ساخته شده است از فرامین فوق استفاده می­کنیم اما اما اما زمانی که خود کلاس را داریم برای گرفتن نام آن از \_\_name\_\_ خالی استفاده باید کرد و اگر کلاس نمونه ای گرفته شده باشد از همان \_\_name\_\_ ولی اگر از آن شی ساخته شده باشد از روش بالا استفاده می شود. نمونه

class bj():

def asf():

print('asdf')

a=bj()

print(bj.\_\_name\_\_)

print((a.\_\_class\_\_.\_\_name\_\_))

* پارامترهایی که برای کلیه اشیاء درون یک ساب پکیج باید ثابت باشد درون یک کلاس والد بنام \_GlobalParameters تعریف می­شوند و تمامی اشیاء درون ساب پکیج مذکور باید از شی فوق ارث بری داشته باشند.
* اشیاء درون ساب پکیج PlantUnits باید دارای یک تابع \_isSimilar باشند که یک شی ورودی میگیرد و این تابع نشان می­دهد که آیا شی دریافتی مشابه با شی جاری می­باشد. این مشابهت درون بخش تحلیل مرحله 0 جهت تعیین احتمال آسیب بکار می­رود.