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

Guide masters:

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

# Introduction

In this Guideline the procedure of modelling and usage of the program has been described for users and steps of modelling has been explained.

Table of Contents

[1- Introduction 2](#_Toc126182625)

[2- Modeling Steps Diagram 6](#_Toc126182626)

[3- Initial Required Knowledge 7](#_Toc126182627)

[3-1- How to install and call the package 7](#_Toc126182628)

[3-2- System of the units 7](#_Toc126182629)

[3-3- Tags 7](#_Toc126182630)

[4- Miscellaneous Commands 8](#_Toc126182631)

[4-1- opr.wipe() 8](#_Toc126182632)

[4-2- opr.wipeAnalysis() 8](#_Toc126182633)

[4-3- opr.warning(string) 8](#_Toc126182634)

[4-4- opr.warningClear() 8](#_Toc126182635)

[4-5- opr.Misc.recorder(filename, SaveStep) 8](#_Toc126182636)

[4-6- opr.Misc. recordedScenarios (filename='') 8](#_Toc126182637)

[4-7- opr.Misc.recordedObjectsClear (filename='') 8](#_Toc126182638)

[4-8- opr.Misc.load1RecordedScenario(ScenarioTag, filename='') 9](#_Toc126182639)

[4-9- opr.Misc.loadScenarioBank(filename) 9](#_Toc126182640)

[4-10- opr.Misc.load1ScenarioOfBank(ScenarioTag) 9](#_Toc126182641)

[4-11- opr.Misc.ClearScenarioBank() 9](#_Toc126182642)

[5- Modeling commands and their usage 10](#_Toc126182643)

[5-1- Hazard Subpackage 10](#_Toc126182644)

[5-1-1- Earthquake 10](#_Toc126182645)

[5-2- DateAndTime Subpackage 10](#_Toc126182646)

[5-2-1- DateTime 10](#_Toc126182647)

[5-3- WindData Subpackage 11](#_Toc126182648)

[5-3-1- WindRose 11](#_Toc126182649)

[5-4- Sites Subpackage 13](#_Toc126182650)

[5-4-1- Site 13](#_Toc126182651)

[5-5- Substance Subpackage 13](#_Toc126182652)

[5-5-1- Material 14](#_Toc126182653)

[5-5-2- DataBank 15](#_Toc126182654)

[5-6- Fragilities Subpackage 15](#_Toc126182655)

[5-6-1- Fragility 15](#_Toc126182656)

[5-6-2- Probit 16](#_Toc126182657)

[5-7- OutFlowModel Subpackage 17](#_Toc126182658)

[5-7-1- TankSimultanious 17](#_Toc126182659)

[5-7-2- TankHole Series 17](#_Toc126182660)

[5-7-3- GasUnitHole 18](#_Toc126182661)

[5-7-4- Liquid10min 18](#_Toc126182662)

[5-7-5- NoOutFlow 18](#_Toc126182663)

[5-8- Connectors Subpackage 18](#_Toc126182664)

[5-8-1- DS\_LOC 19](#_Toc126182665)

[5-8-2- Out\_Physic 19](#_Toc126182666)

[5-8-3- Pb\_LOC 20](#_Toc126182667)

[5-9- DispersionSpreadModels Subpackage 21](#_Toc126182668)

[5-9-1- LiquidSpread 21](#_Toc126182669)

[5-9-2- BritterMcQuaid 22](#_Toc126182670)

[5-9-3- GasGaussian 22](#_Toc126182671)

[5-9-4- LqdSprdGaussianGasDisp 23](#_Toc126182672)

[5-10- PhysicalEffect Subpackage 24](#_Toc126182673)

[5-10-1- fire\_point\_source 24](#_Toc126182674)

[5-10-2- VCE\_TNT 25](#_Toc126182675)

[5-10-3- Safe 25](#_Toc126182676)

[5-11- Safety Subpackage 26](#_Toc126182677)

[5-11-1- Dike 26](#_Toc126182678)

[5-12- PlantUnits Subpackage 26](#_Toc126182679)

[5-12-1- ONGStorage 26](#_Toc126182680)

[5-13- Recorders SubPackage 28](#_Toc126182681)

[5-13-1- Objs\_recorder 29](#_Toc126182682)

[5-13-2- Objs\_recorder\_loader 29](#_Toc126182683)

[5-13-3- Objs\_recorder\_loader (Bank method) 30](#_Toc126182684)

[5-13-4- Recorder 31](#_Toc126182685)

[5-14- Analyze Subpackage 32](#_Toc126182686)

[5-14-1- UniAnalyze 32](#_Toc126182687)

[5-14-2- MultiAnalysis 32](#_Toc126182688)

[5-14-3- MultiParallel 32](#_Toc126182689)

[6- Post Processing 34](#_Toc126182690)

[6-1- Get access to defined objects 34](#_Toc126182691)

[6-2- Subpackages callable global parameters 34](#_Toc126182692)

[6-2-1- Hazard 34](#_Toc126182693)

[6-2-2- DateAndTime 34](#_Toc126182694)

[6-2-3- WindData 35](#_Toc126182695)

[6-2-4- Sites 35](#_Toc126182696)

[6-2-5- Substance 35](#_Toc126182697)

[6-2-6- Fragilities 35](#_Toc126182698)

[6-2-7- OutFlowModel 35](#_Toc126182699)

[6-2-8- DispersionSpreadModels 35](#_Toc126182700)

[6-2-9- PhysicalEffect 36](#_Toc126182701)

[6-2-10- Connectors 37](#_Toc126182702)

[6-2-11- Safety 37](#_Toc126182703)

[6-2-12- PlantUnits 37](#_Toc126182704)

[6-3- PostProcess Subpackage 37](#_Toc126182705)

[6-3-1- ObjsRecorderPP 37](#_Toc126182706)

[6-3-2- RecorderPP 40](#_Toc126182707)

[6-3-3- PlotPP 40](#_Toc126182708)

[6-4- Plot Subpackage 41](#_Toc126182709)

[7- References 42](#_Toc126182710)

# Modeling Steps Diagram

In this part steps of modeling are described and for a complete modelling user should pass the following steps to finish the model. For each step user can uses Python coding commands and features (loops commands, dictionaries, lists …) to make modeling more convenient and it depends on the user. But before start of the analysis, the required objects shown in the following path should be defined.

# Initial Required Knowledge

## How to install and call the package

The package is based on Python programming language and it has published on pypi and users can install this package using following command for windows:

pip install OpenSRANE

For other systems (Mac, Linux …) or platforms using corresponding commands for installing Python packages users can install the package.

By installing the package on the system, users can call it like all the other Python packages or libraries using import command:

import OpenSRANE as opr

## System of the units

OpenSRANE units is according SI and users should enter any value in any part of the code according SI unit’s system.

## Tags

OpenSRANE uses tag numbers to distinguish defined objects in each subpackage. Tag numbers are values that be assigned to the defined objects by user and are unique values for each subpackage and also users can refer to the defined objects using these tag numbers. So users should pay attention that enter none repetitive values for tag numbers for each subpackage objects.

# Miscellaneous Commands

## opr.wipe()

This command cause all OpenSRANE created objects will be delete and it is highly recommended that in the start of any modeling, first use this command.

## opr.wipeAnalysis()

By this command all of the saved results and selected and assigned objects will be clear. Many objects depend on the user definition and generated random values will be select and will be assign to other objects in each simulation. For example, a defined outflow object may be assigned in a simulation to a plant unit object but in next simulation maybe no outflow will be happen, so by wipeAnalysis() command without clearing main objects, only resulted and assigned secondary objects will be clear and the main initial objects will remain.

## opr.warning(string)

## opr.warningClear()

By running the analysis, all the warnings will be appended to Warnings.wrn file. By warningClear() command all recorded warnings will be clear from Warnings.wrn file.

# Modeling commands and their usage

## Hazard Subpackage

In this package the natural hazard intensity and corresponding probability of occurrence defines.

### Earthquake

Obviously is used for defining hazard of earthquake events.

* Command structure:

opr.Hazard.Earthquake(Tag, DefType='PGA', Magnitude=[], Probabilities=[])

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + DefType: Specifies the type of the earthquake magnitude. Currently it has no effect on the program process and the input is only as a string.
  + Magnitude: list of the magnitude values.
  + Probabilities: list of the probability values corresponding to the magnitude values.
* Hypothesizes: Probability values should start from 1 to zero
* Example:

#Generate a Seismic Hazard Object

PGA**=[**x**/**2 **for** x **in** **range(**41**)** **]**

Prob**=[(**40**-**x**)\*\*(**1**/**3**)/**40**\*\*(**1**/**3**)** **for** x **in** **range(**41**)]**

opr.Hazard**.**Earthquake**(**1**,**'PGA'**,**PGA**,**Prob**)** #Create Hazard Object with Tag=1 that is 0th Object

## DateAndTime Subpackage

In this package modules that defines day night ratio and uses for sampling time of the day.

### DateTime

To define day night ratio.

* Command structure:

opr.DateAndTime**.**DateTime**(**Tag**,** Day\_Night\_Ratio**=**2**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Day\_Night\_Ratio: Specifies the day to night ratio. The predefined value is 2 means that in a day 16 hours is day time and 8 hours is night duration.
* Example:

opr.DateAndTime**.**DateTime**(**1**,**Day\_Night\_Ratio**=**2**)**

## WindData Subpackage

Using this package wind model data will be defined by the user. Currently WindRose module is available for this purpose.

### WindRose

This module contains several inputs that can directly be inserted among object definition that is tough and is not suggested.

* Command structure:

opr.WindData**.**WindRose**(**Tag**,** WindDayClassList**=None,** WindNightClassList**=None,** AlphaCOEFlist**=None,** DayWindSpeedList**=None,** DayWindFreqMatrix**=None,** NightWindSpeedList**=None,** NightWindFreqMatrix**=None)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + WindDayClassList: Specifies list of the wind classes corresponding to each wind speed range in windSpeedList in day.
  + WindNightClassList: Specifies list of the wind classes corresponding to each wind speed range in windSpeedList in night.
  + AlphaCOEFlist: Specifies list of the wind height alpha coefficient corresponding to each wind speed range in windSpeedList.
  + DayWindSpeedList: List of wind speed corresponding to each direction for day.
  + NightWindSpeedList: List of wind speed corresponding to each direction for night.
  + DayWindFreqMatrix: A list of wind speed probabilities that for each direction divided to the probabilities of corresponding wind speed range defined in wind list for day.
  + NightWindFreqMatrix: A list of wind speed probabilities that for each direction divided to the probabilities of corresponding wind speed range defined in wind list for night.
* Hypothesizes: Attention that the lists in WindFreqMatrix considered for each direction and the first list is for north direction and consider as zero degree.
* Example:

As said, defining all above parameters while defining the wind rose object is tough and author suggests that define step by step as shown in the following. The below values are according to the example of chapter 7 of Casal book [1] :

#Define Wind Rose and store the wind rose object in a variable

windobj**=**opr.WindData**.**WindRose**(**1**)**

#define wind classes

windobj**.**WindDayClassList**=[**'F'**,**'D'**,**'B'**,**'E'**,**'D'**,**'D'**]**

windobj**.**WindNightClassList**=[**'F'**,**'D'**,**'B'**,**'E'**,**'D'**,**'D'**]**

#define AlphaCOEFlist

windobj**.**AlphaCOEFlist**=[**0.6**,**0.25**,**0.15**,**0.4**,**0.25**,**0.25**]**

#define Wind speeds list

windobj**.**DayWindSpeedList**=[**

**[**1**,**2**],[**2**,**3**],[**3**,**5**],[**5**,**7**],[**7**,**9**],[**9**]]**

windobj**.**NightWindSpeedList**=[**

**[**1**,**2**],[**2**,**3**],[**3**,**5**],[**5**,**7**],[**7**,**9**],[**9**]]**

#Define wind probabilities for Day and night for each direction and corresponding to the wind speeds

windobj**.**DayWindFreqMatrix**=[**

**[**0.446**,**0.372**,**0.355**,**0.109**,**0.017**,**0**],**

**[**0.44**,**0.938**,**1.55**,**0.755**,**0.097**,**0.029**],**

**[**0.898**,**1.321**,**3.06**,**1.402**,**0.767**,**0.892**],**

**[**0.875**,**1.241**,**2.626**,**1.51**,**0.892**,**0.646**],**

**[**0.801**,**0.927**,**1.63**,**0.658**,**0.355**,**0.097**],**

**[**0.87**,**1.121**,**0.984**,**0.309**,**0.023**,**0.029**],**

**[**0.778**,**0.801**,**0.91**,**0.315**,**0.029**,**0**],**

**[**0.652**,**0.875**,**1.35**,**0.498**,**0.086**,**0.023**],**

**[**0.566**,**0.887**,**1.659**,**0.709**,**0.149**,**0**],**

**[**0.583**,**0.807**,**2.128**,**2.998**,**1.041**,**0.137**],**

**[**0.898**,**1.093**,**2.408**,**2.059**,**1.327**,**0.154**],**

**[**1.985**,**2.088**,**2.488**,**1.098**,**0.332**,**0.069**],**

**[**4.067**,**3.123**,**1.442**,**0.292**,**0.063**,**0.011**],**

**[**3.93**,**5.372**,**3.85**,**1.201**,**0.349**,**0.057**],**

**[**1.71**,**1.619**,**2.38**,**0.767**,**0.109**,**0.006**],**

**[**0.698**,**0.469**,**0.383**,**0.154**,**0.011**,**0**],]**

windobj**.**NightWindFreqMatrix**=**windobj**.**DayWindFreqMatrix

## Sites Subpackage

In this subpackage site object will be defined by the user and properties of the site also should be define here. User can define more than one site object but currently only first object is considered for the model.

### Site

This module is used for defining various properties of the site that target plant located in.

* Command structure:

opr.Sites**.**Site**(**Tag**,** Temprature**=**293**,** Pressure**=**100000**,** XSiteBoundary**=[**0**],** YSiteBoundary**=[**0**],** g**=**9.81**,** OngroundTemprature**=None,** Airdensity**=**1.21**,** Humidity**=**0.6**,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Temperature: Specifies site atmosphere temperature.
  + Pressure: Specifies site atmosphere pressure.
  + XsiteBoundary and YSiteBoundary: Specifies the boundary coordinate of the site.
  + g: Earth gravity acceleration.
  + OnGroundTemperature: Specifies the site onground temperature.
  + Airdensity: Obvious.
  + Humidity: Obvious.
* Hypothesizes: Nothing special.
* Example:

opr.Sites**.**Site**(**SiteTAg**,** Temprature**=**25**+**273**,** Pressure**=**1**\***10**\*\***5**,** XSiteBoundary**=[**0**,**100**,**100**,**0**],** YSiteBoundary**=[**0**,**0**,**100**,**100**],** g**=**9.81**)**

## Substance Subpackage

In this subpackage the materials and their properties are defined here by user. There are many properties that can be defined for the substances but all of them are not necessary for the models and depend on the models that user selected for the model, parameters or properties are needed. Most of the needed properties will be request in the warning file if are not defined by the user.

Also, there is DataBank module with predefined materials that can be complete more simply in the future.

### Material

This module is used for defining the substances.

* Command structure:

opr.Substance**.**Material**(**Tag**,** name**=**'No name'**,** Density**=None,** GasDensity**=None,** LiquidPhaseSpreadModelTag**=None,** GasPhaseDispModelTag**=None,** Dynamic\_Viscousity**=None,** Molar\_Heat\_of\_Combustion**=None,** Stoichiometric\_Concentration**=None,** Vapour\_Density**=None,** Volumetric\_Heat\_Capacity**=None,**

Molecular\_Weight**=None,** Molar\_Volume**=None,** Boiling\_Point**=None,** Critical\_Pressure**=None,** Critical\_Temperature**=None,** Melting\_Point**=None,** Standard\_Enthalpy\_of\_Formation**=None,**

Vapour\_Pressure**=None,** Molar\_Enthalpy\_of\_Vaporization**=None,** Specific\_Heat\_of\_Vaporization**=None,** Molar\_Heat\_Capacity**=None,** Specific\_Heat\_Capacity**=None,**

Specific\_Heat\_Ratio**=None,** Autoignition\_Temperature**=None,** Flash\_Point**=None,** Specific\_Heat\_of\_Combustion**=None,** Lower\_Flammability\_Limit**=None,** Upper\_Flammability\_Limit**=None,**

Bioconcentration\_Factor**=None,** Liquid\_Partial\_Pressure\_in\_Atmosphere**=None,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + name: Obvious.
  + Other parameters are obvious from their names and there is no need to more describtion.
* Hypothesizes: Nothing special.
* Example:

opr.Substance**.**Material**(**Tag**=**1**,**name**=**'None'**,**Density**=**250**,** Specific\_Heat\_of\_Combustion**=**45.334**\***10**\*\***6**)**

opr.Substance**.**Material**(**2**,**name**=**'Imaginary Material'**,** Density**=**123**,**Vapour\_Density**=**125**,** Specific\_Heat\_of\_Combustion**=**45.334**\***10**\*\***6**,)**

### DataBank

This module is used for defining the substances using predefined data.

* Command structure:

opr.Substance**.**DataBank**.**SAVEDMATERIALNAME(Tag)

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
* Hypothesizes: Nothing special.
* Example:

opr.Substance**.**DataBank**.**Butene**(**3**)**

opr.Substance**.**DataBank**.**Dimethylhydrazine**(**4**)**

## Fragilities Subpackage

Fragilities and Probits models are defined using this part. Because same nature of these models, they are considered in a subpackage.

### Fragility

This module is used to define the structures fragility under natural magnitudes that defined using Hazard subpackage. So it is obvious that one axis of the fragility should be the magnitude that were considered for the defined hazard object.

* Command structure:

opr.Fragilities**.**Fragility**(** Tag**,** modename**=**'No Fragility Mode name'**,** Distribution\_Type**=**'normal'**,** mean**=**1**,** StdDev**=None,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Modename: Name of the corresponding mode that fragility is being define.
  + Distribution\_Type: Type of the distribution that is considered for the fragility. Currently there is three types that can be select by the users: 'normal', 'lognormal', 'constant'.
  + Mean: mean of the distribution.
  + StdDev: Standard deviation of the distribution.
* Hypothesizes: Nothing special.
* Example:

opr.Fragilities**.**Fragility**(**Tag**=**1**,**Distribution\_Type**=**'normal'**,**modename**=**'EBF'**,**mean**=**15**,**StdDev**=**2.5**)**

opr.Fragilities**.**Fragility**(**Tag**=**2**,**Distribution\_Type**=**'lognormal'**,**modename**=**'Gas Failure'**,**mean**=**15**,**StdDev**=**4**)**

### Probit

This module is used to define the structures vulnerability to the physical effects. Obviously one of its axes is the magnitude of the physical effect (Radiation, Overpressure, Toxic) that this model is considered for.

* Command structure:

opr.Fragilities**.**Probit**(** Tag**,** Distribution\_Type**='normal',** K1**=1,** K2**=0.5,** Scale\_Factor**=1,** MinRndVar**=0)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Distribution\_Type: Type of the distribution that is considered for the fragility. Currently there is three types that can be select by the users: 'normal', 'lognormal'.
  + K1, K2: Probit Normal () and Lognormal () coefficients.
  + Scale\_Factor: Scale factor for magnitude of the physical effect that converts the Probit magnitude values to the SI system.
  + MinRndVar: A value that will be consider as the minimum intensity of random variable and the defined probit function returns 0 for random variables less than this value (Example: if minimum radiation that injure is 4 this value should be enter 4 then for any value less or equal to 4 its probit will return 0 as the probit probability)
* Hypothesizes: Nothing special.
* Example:

Radiation**=**5

opr.Fragilities**.**Probit**(**Tag**=**Radiation**,** Distribution\_Type**=**'lognormal'**,** K1**=**2**,** K2**=-**30.9**,**Scale\_Factor**=**1000**)**

OverPressure**=**6

opr.Fragilities**.**Probit**(**Tag**=**OverPressure**,** Distribution\_Type**=**'normal'**,** K1**=**1.37**,** K2**=-**1.47**,**Scale\_Factor**=**1000**)**

Toxic**=**7

opr.Fragilities**.**Probit**(**Tag**=**Toxic**,** Distribution\_Type**=**'lognormal'**,** K1**=**0.71**,** K2**=-**9.82**)**

## OutFlowModel Subpackage

In this part, Outflow models should be defined for the program. For each model, depend the type of the model and material user should select proper corresponding material and Outflow model settings. In the following current available models will be described.

### TankSimultanious

This model considers a simultaneous release from Tank with a volume equal to the entered ratio of total amount of stored material by the user.

* Command structure:

opr.OutFlowModel**.**TankSimultanious**(**Tag**,** Release\_Ratio**=1)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Release\_Ratio: A value between 0 and 1 that defines the ratio of released material with respect to the total amount of the stored material in the Tank. As it is shown in the command structure if user don't enter anything for this parameter, it will be consider equal unity.
* Hypothesizes: Nothing special.
* Example:

opr.OutFlowModel**.**TankSimultanious**(**Tag**=**9**,** Release\_Ratio**=**0.65**)**

### TankHole Series

4 models that start with TankHole (TankHole, TankHoleInitRate, TankHoleDuration, TankHoleFixStep) models **Liquids** outflow from a hole in the body of the tank with small changes to each other. TankHole models the outflow until the outflow volume become equal to the Tank containment volume, TankHoleInitRate is same TankHole model with outflow rate equal to the initial rate, TankHoleDuration is same Tankhole model but its calculations will be stop when the release duration become equal to defined duration and TankHoleFixStep is same TankHole but with limited steps of calculations. Models are according [1].

* Command structure:

opr.OutFlowModel**.**TankHole**(**Tag**,** Hole\_Diameter**=0.01,**

Hole\_Height\_FromBot**=0,** delta\_t**=0.1,** Cd**=1,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Hole\_Diameter : Obvious.
  + Hole\_Height\_FromBot: Obvious.
  + delta\_t: Calculation steps time interval.
  + Cd: Discharge coefficient .
  + Other Parameters in different models are obvious.
* Hypothesizes: Nothing special.
* Example:

opr.OutFlowModel**.**TankHole**(**Tag**,** Hole\_Diameter**=**0.05**,** Hole\_Height\_FromBot**=**0**,** delta\_t**=**500**,** Cd**=**1**)**

opr.OutFlowModel**.**TankHoleInitRate**(**2**,** Hole\_Diameter**=**0.01**,** Hole\_Height\_FromBot**=**1**,** delta\_t**=**500**,** Cd**=**0.62**)**

opr.OutFlowModel**.**TankHoleDuration**(**3**,** Hole\_Diameter**=**0.5**,** Hole\_Height\_FromBot**=**0.5**,** delta\_t**=**500**,** Cd**=**0.62**,**Duration**=**3000**)**

opr.OutFlowModel**.**TankHoleFixStep**(**4**,** Hole\_Diameter**=**0.5**,** Hole\_Height\_FromBot**=**0.5**,** delta\_t**=**500**,** Cd**=**0.62**,**StepNumber**=**30**)**

### GasUnitHole

Model for **gas** outflow

### Liquid10min

A very simple model that is not practical and is only for

### NoOutFlow

## Connectors Subpackage

Connectors are used to connect some models to their following models with a probability distribution. It is assumed that there more than one following event that could happen after current event, so by this type of definition user can define one or more than one following model that increase the flexibility of the program.

**Important Point:** Users should attention that defined tags for all objects or models of this subpackage should be unique. For example, if user defined DS\_LOC connectors with tags 1 and 2 then to define 1-1-1- Out\_Physic connectors, the tags should be continued from 3.

### DS\_LOC

This command connects a fragility model (Damage state) to its corresponding outflow models and user for each damage state or fragility defines a list of probable outflow models with their corresponding probability and if during the analysis a damage under a fragility happens, then code uses the fragility's outflow models to calculate the following out flow.

Such approach has been used to connect probit functions to outflow models and also to connect outflow models to physical events that will be described in the following.

* Command structure:

opr.Connectors**.**DS\_LOC**(** tag**,** FragilityTag**,** OutFlowModelTagList**,** LOCProbabilityList**=None,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + FragilityTag: The defined fragility tag that user wants to define its corresponding outflow models.
  + OutFlowModelTagList: List of outflow models that can happen after happening defined fragility tag.
  + LOCProbabilityList: List of the outflow models probabilities that program select a loss of containment (LOC) model according them.
* Hypothesizes: If user do not enter LOCProbabilityList or enter None, the program will consider a uniform distribution for the defined models. If summation of the defined probabilities does not be equal to unity, program will normal the according their weights to be equal unity.
* Example:

opr.Connectors**.**DS\_LOC**(**1**,**FragilityTag**=**1**,**OutFlowModelTagList**=[**1**,**2**,**3**,**8**],**LOCProbabilityList**=[**4**,**3**,**2**,**1**])**

opr.Connectors**.**DS\_LOC**(**2**,**FragilityTag**=**2**,**OutFlowModelTagList**=[**6**,**7**],**LOCProbabilityList**=[**1**,**1**])**

### Out\_Physic

Same definition of DS\_LOC but for connecting each outflow model and material or substance to their corresponding physical effects.

* Command structure:

opr.Connectors**.**Out\_Physic**(** Tag**,** OutFlowTag**,** MaterialsTagList**,** PhysicalEffectTagList**,** PhysProbabilityList**=None,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + OutFlowTag: The defined outflow model tag that user wants to define its corresponding physical effect models.
  + MaterialsTagList: List of defined materials that user wants to their corresponding outflow and physical effect.
  + PhysicalEffectTagList: List of physical effect models that can happen after happening defined OutFlowTag tag.
  + PhysProbabilityList: List of the outflow models probabilities that program select a loss of containment (LOC) model according them.
* Hypothesizes: If user do not enter PhysProbabilityList or enter None, the program will consider a uniform distribution for the defined models. If summation of the defined probabilities does not be equal to unity, program will normal the according their weights to be equal unity.
* Example:

opr.Connectors**.**Out\_Physic**(**Tag**=**10**,**OutFlowTag**=**5**,** MaterialsTagList**=[**1,2,3**],** PhysicalEffectTagList**=[**3**],**PhysProbabilityList**=[**1**])**

opr.Connectors**.**Out\_Physic**(**Tag**=**11**,**OutFlowTag**=**6**,** MaterialsTagList**=[**4**],** PhysicalEffectTagList**=[**3**],**PhysProbabilityList**=[**1**])**

### Pb\_LOC

Same definition of DS\_LOC but for connecting each probit model to its corresponding physical effects.

* Command structure:

opr.Connectors**.**Pb\_LOC**(** tag**,** ProbitTag**,**

OutFlowModelTagList**,** LOCProbabilityList**=None,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + ProbitTag: The defined probit model tag that user wants to define its corresponding physical effect models.
  + OutFlowModelTagList: List of outflow models that can happen after happening defined probit tag event.
  + LOCProbabilityList: List of the outflow models probabilities that program select a loss of containment (LOC) model according them.
* Hypothesizes: If user do not enter LOCProbabilityList or enter None, the program will consider a uniform distribution for the defined models. If summation of the defined probabilities does not be equal to unity, program will normal the according their weights to be equal unity.
* Example:

opr.Connectors**.**Pb\_LOC**(**tag**=**13**,** ProbitTag**=**5**,** OutFlowModelTagList**=[**1**,**2**,**3**,**4**,**5**],** LOCProbabilityList**=[**1**,**1**,**1**,**1**,**1**])**

## DispersionSpreadModels Subpackage

In this subpackage the gas dispersion and liquid spread models define by the user. It is possible to for any dispersion type of each material define one dispersion model.

### LiquidSpread

As it is obvious from the name of the module this model is used for the liquid dispersion of the liquid substances. It is obvious that user should consider this material for a liquid substance.

* Command structure:

opr.DispersionSpreadModels**.**LiquidSpread**(**tag**,** MatTags**,** OutFlowModelTags**,** MinDisThickness**=**0.01**,** Surface\_Roughnesslist**=[**0.0001**,** 0.0002**],** Surface\_RoughnessThickness**=[**0.005**,** 0.01**],)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + MatTags: List of defined materials that user wants to consider this module for them as their behaviour.
  + OutFlowModelTags: List of outflow models that this model can happen after happening them.
  + MinDisThickness: Minimum thickness of the liquid until it has not reached to the dike wall (if dike has been defined before). If user do not define any value for this parameter, it will be considered equal to 0.01 m.
  + Surface\_Roughnesslist: List of surfaces roughness’s.
  + Surface\_RoughnessThickness: Thickness values corresponding to each Roughnesslist value.
* Hypothesizes: Nothing special.
* Example:

opr.DispersionSpreadModels**.**LiquidSpread**(**Tag**=**1**,** MatTags**=[**1**,**2**],** OutFlowModelTags**=[**1**,**2**,**3**,**4**],**MinDisThickness**=**0.005**,)**

### BritterMcQuaid

This model is used for gas dispersions and its formulation is according [1].

* Command structure:

opr.DispersionSpreadModels**.**BritterMcQuaid**(**Tag**,** MatTags**,** OutFlowModelTags**,** C0**=1,** MassParts**=20,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + MatTags: List of defined materials that user wants to consider this module for them as their behavior.
  + OutFlowModelTags: List of outflow models that can happen after happening defined fragility tag.
  + C0: Initial concentration of the gas at the outflow point. If user do not define any value for this parameter, it will be considered equal to 1.
  + MassParts: to calculate the outflow mass volume, code uses a numerical approach and divide the volume in the range of the LFL and UFL to MassParts parts and do the calculations according these parts.
* Hypothesizes: Nothing special.
* Example:

opr.DispersionSpreadModels**.**BritterMcQuaid**(**Tag**=**2**,**MatTags**=[**5**],**OutFlowModelTags**=[**6**,**7**],** C0**=**2.84**)**

### GasGaussian

This model is used for gas dispersions and its formulation is but with gaussian algorithm according [1]. It is assumed that the outflow happen from a point and the point height is considered as the height of the outflow point location. The gas concentration will be considered according gas outflow values, so if the outflow model does not consider any outflow values so, this module, do not calculate any dispersion because of no gas outflow.

* Command structure:

GasGaussian**(**tag**,** MatTags**,** OutFlowModelTags**,** OutFlowHeight**=**1**,** GasConstant**=**8.31446261815324**,** GasDispersionXSegments**=**10**,** GasDisperstionErrorPercent**=**1**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + MatTags: List of defined materials that user wants to consider this module for them as their behavior.
  + OutFlowModelTags: List of outflow models that can can cause this type of dispersion.
  + OutFlowHeight: Hight that gas is outflowing.
  + GasDispersionXSegments: number of segments that are considered to calculate the dispersion and also dispersed mass of gas.
  + GasDisperstionErrorPercent: Error percentage for numerical calculations.
* Hypothesizes: Nothing special.
* Example:

opr.DispersionSpreadModels**.**GasGaussian**(**tag**=**1**,**MatTags**=[**1**],** OutFlowModelTags**=[**1**,**2**,**3**,**4**])**

### LqdSprdGaussianGasDisp

This module is for modeling the spread of the liquids and then calculate its gas dispersion in the environment. So, it is appropriate for liquids that their gas dispersions are important to be considered.

* Command structure:

opr.DispersionSpreadModels**.**LqdSprdGaussianGasDisp**(**

tag**,** MatTags**,** OutFlowModelTags**,**

MinDisThickness**=**0.01**,**

Surface\_Roughnesslist**=[],**

Surface\_RoughnessThickness**=[],**

Vaporization\_Delta\_t**=**10**,**

TotalDuration**=**1800**,**

GasConstant**=**8.31446261815324**,**

GasDispersionXSegments**=**10**,**

GasDisperstionErrorPercent**=**1**,)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + MatTags: List of defined materials that user wants to consider this module for them as their behaviour.
  + OutFlowModelTags: List of outflow models that can happen after happening defined fragility tag.
  + MinDisThickness: Minimum thickness of the liquid until it has not reached to the dike wall (if dike has been defined before). If user do not define any value for this parameter, it will be considered equal to 0.01 m.
  + Surface\_Roughnesslist: List of surfaces roughness’s.
  + Surface\_RoughnessThickness: Thickness values corresponding to each Roughnesslist value.
  + Vaporization\_Delta\_t:
  + TotalDuration:
  + GasDispersionXSegments: number of segments that are considered to calculate the dispersion and also dispersed mass of gas
  + GasDisperstionErrorPercent: Error percentage for numerical calculations.
* Hypothesizes: Nothing special.
* Example:

opr.DispersionSpreadModels**.**LqdSprdGaussianGasDisp**(**tag**=**4**,**MatTags**=[**3**,**4**],**OutFlowModelTags**=[**1**,**2**,**3**,**4**],**MinDisThickness**=**0.005**,**Vaporization\_Delta\_t**=**10**,** TotalDuration**=**1800**,**GasConstant**=**8.31446261815324**,**GasDispersionXSegments**=**10**,**GasDisperstionErrorPercent**=**1**,)**

## PhysicalEffect Subpackage

As obvious from the name of subpackage, this package is for modelling the physical events effect.

### fire\_point\_source

This module is for modelling the heat radiation of a burning liquid. If there were no liquid spread this module do not return any results.

* Command structure:

opr.PhysicalEffect**.**fire\_point\_source**(**tag**,** minf**=**0.055**,** k**=**1.5**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + minf: is burning velocity for an infinite diameter pool. some sample values are presented in table 3.8 Casal book.
  + k: constant according table 3.8 Casal book
* Hypothesizes: Nothing special.
* Example:

opr.PhysicalEffect**.**fire\_point\_source**(**1**,**minf**=**0.0501**,**k**=**1.5**)**

### VCE\_TNT

This module is for modelling Vapor Cloud explosion with TNT model. This model doesn’t work if no gas dispersion calculated in corresponding dispersion model in the previous step. So, the defined dispersion model should consider gas dispersion and if it models only liquid spread so no results will be back.

* Command structure:

opr.PhysicalEffect**.**VCE\_TNT**(**tag**,** Etta**=**0.03**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Etta: Explosion Yield Factor That is between 1% to 10% and it is recommended to be considered equal to 3%
* Hypothesizes: Nothing special.
* Example:

opr.PhysicalEffect**.**VCE\_TNT**(**1**)**

### Safe

As obvious from its name, this module case no physical effect even if hazardous material released in environment.

* Command structure:

opr.PhysicalEffect**.**SAFE**(**tag**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
* Hypothesizes: Nothing special.

## Safety Subpackage

This subpackage is for defining the barrier and safety systems. Their properties will be used by other objects that these safety object/s assigned to them.

### Dike

This module is used to define dike objects. By assigning this object to plant units, the outflow area will be limited to the area of defined and assigned dikes.

* Command structure:

opr.Safety**.**Dike**(**tag**,** Height**=None,** Area**=None)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + Height: Hight of the dike.
  + Area: Area is surrounded by the dike.
* Hypothesizes: Nothing special.
* Example:

opr.Safety**.**Dike**(**4**,**2**,**50**\*\***2**)**

## PlantUnits Subpackage

Very Important subpackage that is responsible to define the plant unit objects and assign their properties and defined related objects to them.

### ONGStorage

ON Ground Storage tank object is defined by this module. Defined features also should be assigned by their tags to this/these object/s.

* Command structure:

opr.PlantUnits**.**ONGStorage**(**tag**,**

SiteTag**=None,**

DikeTag**=None,**

SubstanceTag**=None,**

FragilityTagNumbers**=None,**

Horizontal\_localPosition**=**0**,**

Vertical\_localPosition**=**0**,**

Surface\_Roughness**=None,**

Pressure**=**0**,**

Temperature**=**0**,**

SubstanceVolumeRatio**=None,**

Diameter**=None,**

Height**=None,**

GroundTemperature**=None,**

Ks\_Soil\_Thermal\_conductivity**=None,**

Alphas\_Soil\_thermal\_diffusivity**=None,**

boundary\_points\_Number**=**20**,**

boundary\_points\_height\_levels**=**10**,**

pressure\_probit\_tag**=None,**

radiation\_probit\_tag**=None,**

RadiationDifferenceDose=1000**,**

**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + SiteTag: Tag of the site.
  + DikeTag: Tag of the dike that surrounded the tank. There is no need to define if the tank is not surrounded by any dike.
  + SubstanceTag: Tag of the tank substance.
  + FragilityTagNumbers: List of the tank fragilities tag.
  + Horizontal\_localPosition: Horizontal location of the tank respect to the local reference point (0,0).
  + Vertical \_localPosition: Vertical location of the tank respect to the local reference point (0,0).
  + Surface\_Roughness: Value of the surface roughness. This value will be used for liquid dispersion models.
  + Pressure: Internal pressure value of the tank.
  + Temperature: Internal temperature value of the tank.
  + SubstanceVolumeRatio: Ratio of the tank content to the total volume of the tank.
  + Diameter: Tank diameter.
  + Height: Tank height.
  + GroundTemperature: Temperature of the ground around the tank that will be used for liquids vaporization. If user do not enter any value, it will be considered equal to site temperature.
  + Ks\_Soil\_Thermal\_conductivity: Soil thermal conductivity or Ks that will be used for liquid vaporization calculation. If user do not enter any value, it will be considered equal to 0.9.
  + Alphas\_Soil\_thermal\_diffusivity: Soil thermal diffusivity that will be used for liquid dispersion and If user do not enter any value, it will be considered equal to
  + boundary\_points\_Number: Number of the boundary points that program uses them to calculate the pressure and temperature at these points.
  + boundary\_points\_height\_levels: Number of the levels that boundary points will be repeated in the height of the tank. If boundary points defined equal to 20 and heigh points level defined equal to 10, there will be 200 points on the tank body that temperatures and pressures will be calculated for.
  + pressure\_probit\_tag: Tag of the defined probit object to be use for vulnerability under pressure loads.
  + radiation\_probit\_tag: Tag of the defined probit object to be use for vulnerability under temperature loads.
  + RadiationDifferenceDose: A thermal radiation dose if the increase of radiation become more than this value, code will check the vulnerability of the object using the new thermal radiation dose and defined probit function. In any level it may new units suffer damage and create thermal radiation dose for all other not damaged units, but for far units these doses are less and program shouldn’t recheck the thermal vulnerability. So, if the increasement of the radiation becomes more than RadiationDifferenceDose the code assume that a high new radiation happened and checks the vulnerability of unit under the new total dose.
* Hypothesizes: The default values are shown in the command structure.
* Example:

opr.PlantUnits**.**ONGStorage**(**1**,**SiteTag**=**SiteTAg**,**

DikeTag**=**2**,** Horizontal\_localPosition**=**10**,** Vertical\_localPosition**=**15**,** Pressure**=**2**\***10**\*\***5**,** Temperature**=**273**+**3**,**SubstanceTag**=**1**,**

FragilityTagNumbers**=[**1**,**2**,**4**,**6**],**

radiation\_probit\_tag**=**2**,** pressure\_probit\_tag**=**4**,**

Diameter**=**10**,** Height**=**8**,** SubstanceVolumeRatio**=**0.85**)**

## Recorders SubPackage

The Recorders is a very important subpackage that users should be familiar completely. This subpackage is responsible to record data of each simulation. Currently there are two different modules for this purpose that are described in the following.

### Objs\_recorder

Using this module, a file will be determined by the user to record all simulated scenarios objects. By every analyze, the created objects and results will be record in the mentioned file and user can call them using the load commands.

* Command structure:

opr.Recorders**.**Objs\_recorder**(**

tag**,**

filename**='',**

SaveStep**=100,**

fileAppend**=True,**

**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + filename: Name of the file that user wants to record data in.
  + SaveStep: Number of steps that after that data will be move to the file and memory become empty. Bigger values cause faster analysis but it needs enough system memory.
  + fileAppend: True says that if the filename exists, add the recorded scenarios to the existing file and false will clear the file if exists.
* Hypothesizes: Nothing special.
* Example:

opr.Recorders**.**Objs\_recorder**(**tag**=**1**,** filename**=**'Scenariostest2.dat'**,** SaveStep**=**100**,** fileAppend**=False,)**

### Objs\_recorder\_loader

This command contains the following subcommands that are used to load the recorded data using Objs\_recorder. In the following each subcommand with their usage are described.

Bank method is another way to load scenarios. THIS METHOD USES HUGE AMOUNT OF MEMORY AND NOT RECOMMENDED FOR LARGE SCENARIOS NUMBER. It loads all scenarios into the memory and so calling them are so much fast but if there were no enough memory probably user encounter with system problem!

So just use it when a powerful system is available.

#### loadScenarioBank, load1ScenarioOfBank, ClearScenarioBank

User can send all recorded scenarios (ScenarioBank) to the memory and call them faster. It is obvious that with weak systems with low memory this method may encounter with system hanging. Also, it returns all scenarios as a dictionary that keys are the number of the simulation.

opr.Recorders**.**Objs\_recorder\_loader**.**loadScenarioBank**(**filename**)**

By defining the file name to above command, all the recorded scenarios with all their objects will be load in the memory and using the following command each scenario will be load in the memory very faster than previous method.

opr.Recorders**.**Objs\_recorder\_loader**.**load1ScenarioOfBank**(**ScenarioTag**)**

By the following command memory will be clear from loaded Bank of scenarios.

opr.Recorders**.**Objs\_recorder\_loader**.**ClearScenarioBank**()**

### Recorder

Using this module, a specific result will be recorded for each simulation. The results that can be recorded by this command are listed in the following. The command is fast and it do not take much huge hard disk space. The simulated scenarios will no longer exist and user cannot load them after analysis.

* Command structure:

opr.Recorders**.**recorder**(**

tag**,**

filename**='Recorder.dat',**

fileAppend**=True,**

recordfield**='DamageLevel',**

NodesGroupTag**=1,**

**)**

* Input parameters:
  + Tag: Unique integer value that will be used for referring to the defined elements or objects.
  + filename: Name of the file that user wants to record data in.
  + fileAppend: True says that if the filename exists, add the recorded scenarios to the existing file and false will clear the file if exists.
  + recordfield: Specifies the field of data that want to record. The selections are as the following
    - 'DamageLevel': Records the damage level of each plant unit with respect to the tag number (DamageLevel: the level that the plant unit got damage)
    - 'NodesGroupIsDamaged': Records the nodesgroup with tag equal NodesGroupTag is damaged or not (0 for not damaged and 1 for damaged or failed or dead)
    - 'FragilityTag': Records Fragility tag of each damaged plant unit. Every recorded value is the fragility tag number that cause damage to the corresponding plant unit
    - 'LOC': Records loss of containment value of each plant unit.
    - 'HazardMag': Records the sampled hazard magnitude.
    - 'NodesRadiationOverPressure': Records NodesGroup Radiation and OverPressure values (To prevent creating huge files, the number are recorded with 4 decimals)
    - 'NodesToxic': Records the NodesGroup toxic value
    - 'NodesRadiationProbit': Records the NodesGroup Radiation probit value [Probit(Radiation)]
    - 'NodesOverPressureProbit': Records the NodesGroup OverPressure probit value [Probit(OverPressure)]
  + NodesGroupTag: if a property of a NodesGroup is considered in the recordfield to record, the tag of the NodesGroup object should be define here.
* Hypothesizes: Nothing special.
* Example:

opr.Recorders**.**recorder**(**tag**=**1**,** filename**=**'Recorder.dat'**,** fileAppend**=False,** recordfield**=**'DamageLevel'**,)**

Results will be same in text format in the defined filename. Using the following command, the recorded results also can be call as a dictionary by the following command. The keys are the scenario number and the corresponding values are the specified recorded field.

opr.Recorders**.**recorder**.**LoadRecorderfile**(** filename**='')**

## Analyze Subpackage

This subpackage contains commands that run the analyze for the user defined model. Obviously this subpackage commands should be used at the end of the modeling commands.

There are currently two commands for this purpose that described in the following.

### UniAnalyze

By running this command, one scenario analysis will be run and the defined algorithm will be implemented for defined model. The main structure of the command has been shown in the following.

opr.Analyze**.**ScenarioAnalyze**.**UniAnalyze**(**SavetoFile**=True)**

When analyze finished, all objects and results are in the memory and user can use them for post processing. SavetoFile option cause that the resulted scenario will be remain in the memory and when the number of scenarios in the memory become equal recorder object SaveStep number, they will be save into the reorder file specified in the recorder object.

### MultiAnalysis

This command as obvious from its name, implement multiple analysis equal to specified AnalysisNumber.

opr.Analyze**.**ScenarioAnalyze**.**MultiAnalysis**(**AnalysisNumber**=100)**

When the number of analyses reaches to the defined SaveStep number, the resulted scenarios will be saved in the defined file specified in the recorder object.

### MultiParallel

By this command users can analyze their model with multiple cores or CPUs. Depend on the number of the cores the analysis duration reduced more.

opr.Analyze**.**ScenarioAnalyze**.**MultiParallel**(**

AnalysisNumber**=100,** NumberOfProcessors**=3,)**

If user define NumberOfProcessors more than the number of the available cores, it will be reduced to the available number of the cores.

Users should pay attention that this command should be used after (if \_\_name\_\_=='\_\_main\_\_':) and outside of that, it will be encounter with error!. Because of mentioned limitation it cannot be used inside the Jupyter NoteBook.

Here is an example of how should it be hire:

**if** \_\_name\_\_**==**'\_\_main\_\_'**:**

opr.Analyze**.**ScenarioAnalyze**.**MultiParallel**(**AnalysisNumber**=**1000000**,**NumberOfProcessors**=**15**)**

# Post Processing

By finishing analysis (or maybe at the end of each analysis) users, need resulted data and values to export various results from them. Also, there is a need of plotting model and checking the results visually. In this chapter getting access to the defined objects and available parameters and results that can be call are described. Also, provided subpackage for plotting are described.

## Get access to defined objects

In any subpackage there is an *ObjManager* class that can be call and have 2 variables that stores defined objects data (*Taglst, TagObjDict*).

* *Taglst:* Returns list of the defined objects tags.
* *TagObjDict:* Returns dictionary of the defined objects that its keys are tag of the objects and its values are the corresponding defined object.

So, users to get access to the defined object using defined tag can use *TagObjDict* and enter the tag of the object and get the defined object. For example, if user want to get defined plant unit object with tag 4:

opr.PlantUnits.ObjManager.TagObjDict[4]

or if user wants to get all defined tags can get them using *Taglst:*

opr.PlantUnits.ObjManager.Taglst

## Subpackages callable global parameters

For any defined object after finishing the analysis, user can call the parameters that that are evaluated. Each object has global parameters and also its internal defined parameters. In this part global parameters of each object that users can call and check the resulted values are described.

### Hazard

* *.SampledMagnitude*

Example:

opr.Hazard.ObjManager.TagObjDict[1].SampledMagnitude

### DateAndTime

* *.Day\_Night\_Ratio* (Defined by user)
* *.SampledisDay*

Example:

opr.DateAndTime.ObjManager.TagObjDict[1].SampledisDay

### WindData

* *.WindClass*
* *.WindDirection*
* *.WindSpeed*
* *.AlphaCOEF*
* *.isCalmn*

Example:

opr.WindData.ObjManager.TagObjDict[1].WindSpeed

### Sites

Nothing except what are defined by the user.

### Substance

Nothing except what are defined by the user.

### Fragilities

Nothing except what are defined by the user.

### OutFlowModel

* *.t\_release* : Returns list of the release time
* *.MassLiquidReleaseRate* : Returns list of Liquid release rate at each t\_release
* *.dMassLiquid\_release* : Returns list of released mass of the liquid at each t\_release
* *.TotalMassLiquid\_Release* : Returns list of total released mass at each t\_release
* *.MassGasReleaseRate* : Returns list of Gas mass release rate at each t\_release
* *.dMassGas\_release* : Returns list of released mass of the gas at each t\_release
* *.TotalMassGas\_Release* : Returns list of total released mass of gas at each t\_release

Example: To get list of total gas released of the PlantUnit with tag 1

opr.PlantUnits.ObjManager.TagObjDict[1].OutFlowModelObject.TotalMassGas\_Release

### DispersionSpreadModels

* *.t\_disp* : Returns list of the Dispersion time
* *.LiquidRadious* : Returns list of the liquid radius at each t\_disp
* *.LiquidCenter* : Returns list of the liquid center at each t\_disp
* *.LiquidThickness* : Returns list of the liquid thickness at each t\_disp
* *.t\_dispLiquidVaporization* : Returns list of the liquid vaporization time.
* *.LiquidVaporizationMassRate* : Returns list of the vaporization mass rate at each t\_dispLiquidVaporization
* *.LiquidVaporizationMass* : Returns list of the total vaporized mass at each t\_dispLiquidVaporization
* *.GasExplosiveMass* : Returns list of the total explosive mass at each t\_dispLiquidVaporization
* *.GasExplosiveCenterX* : Returns list of the explosive mass x center at each t\_dispLiquidVaporization
* *.GasExplosiveCenterY* : Returns list of the explosive mass y center at each t\_dispLiquidVaporization
* *.GasExplosiveCenterZ* : Returns list of the explosive mass z center at each

Example: To get list of total mass of vaporized gas of the PlantUnit with tag 1

opr.PlantUnits.ObjManager.TagObjDict[1].DispersionSpreadModelObject.GasExplosiveMass

### PhysicalEffect

* *.Thermal\_Radiation\_at\_Point(x,y,z)* : Returns thermal radiation and entered point,
* *.RadiationBoundary(Radiation,Height,PointNumber)* : Returns Boundary points with equal radiation value equal to entered Radiation and entered height.
* *.OverPressure\_at\_Point(x,y,z)* : Returns over pressure value at entered point.
* *.OverPressureBoundary(OverPressure, Height, PointNumber)* : Returns Boundary points with equal Over pressure value equal to entered Radiation and entered height.

Example: To get Thermal radiation value of a fired PlantUnit with tag 1 at point with coordinate (5,8,2):

opr.PlantUnits.ObjManager.Objlst[1].PhysicalEffectObject.Thermal\_Radiation\_at\_Point(5,8,2)

### Connectors

Nothing except what are defined by the user.

### Safety

Nothing except what are defined by the user.

### PlantUnits

* *.DamageSource:* Returns damage source name.
* *.DamageSourceTag:* Returns damage source tag.
* *.DamageSourceDose:* Returns damage source dose.
* *.DamageSourceType:* Returns damage source type.
* *.DamageFragilityTag:* Returns Fragility/Probit tag that cause damage.
* *.DamageLevel:* Returns damage level.
* *.OutFlowModelTag:* Returns Outflow model tag.
* *.OutFlowModelname:* Returns Outflow model name.
* *.OutFlowModelObject:* Returns Outflow model object.
* *.DispersionSpreadModelTag:* Returns Dispersion model tag.
* *.DispersionSpreadModelname:* Returns Dispersion model name.
* *.DispersionSpreadModelObject:* Returns Dispersion model Object.
* *.PhysicalEffectModelTag:* Returns Physical effect model tag.
* *.PhysicalEffectModelname:* Returns Physical effect model name.
* *.PhysicalEffectObject:* Returns Physical effect model object.

### NodesGroup

* *.isDamagedList:* Returns list of damage condition of each node
* *.DamageSource:* Returns list of damage source name of each node
* *.DamageSourceTag:* Returns list of damage source tag of each node
* *.DamageSourceDose:* Returns list of damage source dose of each node
* *.DamageSourceType:* Returns list of damage source type of each node
* *.Radiation\_Intensity:* Returns list of Radiation dose of each node
* *.OverPressure\_Intensity:* Returns list of OverPressure dose of each node
* *.Toxic\_Intensity:* Returns list of Toxic dose of each material of each node
* *.Radiation\_Probit:* Returns list of probit value corresponding to the Radiation value at each node [Probit(Radiation)]
* *.OverPressure\_Probit:* Returns list of probit value corresponding to the OverPressure value at each node [Probit(OverPressure)]

## PostProcess Subpackage

In this subpackage, some specific data are exported from recorders. Currently there are ObjsRecorder and Recorder modules for recording. In the following the modules in the PostProcess packages are described.

### ObjsRecorderPP

This module takes the name of the recorded ObjsRecorder filename and returns a dictionary of analyzed resulted data. Example:

Results=opr.PostProcess.ObjsRecorderPP.Analyze('PetrochemicalModel',100)

In the above example 'PetrochemicalModel' is the filename of the ObjsRecoder that recorded all the scenarios of the model and is located on the hard disk. Above analyze returns a dictionary that its keys and values are described in the following.

#### DamagedLevelList

By calling this key, a list of dictionaries will be shown that each dictionary is related to a scenario and shows tag of the elements and level of damage. Example:

Results['DamagedLevelList']

[{1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None}, {1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None}, {1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None},{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 1},{1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None},{1: 0, 2: 0, 3: 3, 4: 4, 5: 1, 6: 0, 7: 1, 8: 2},…]

As it is seen in above sample, for each plant unit tag in each dictionary, the damage level of each plant unit has been shown. For example, in the last one, plants with tag 1,2,6 has damaged in 0 level and plants with tag 5, 7 has damaged in level 1 and unit with tag 3 damaged in level 3 and …

#### FragilityTagList

This key, returns a list of dictionaries that each dictionary is related to a scenario and each key refers to a plant unit tag and the corresponding value shows the tag of defined fragility or probit that cause damage. Example:

Results['FragilityTagList']

[{1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None}, {1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None}, {1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None}, {1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None},{1: 2, 2: 2, 3: 1, 4: 1, 5: 1, 6: 3, 7: 1, 8: 3}, {1: None, 2: None, 3: 1, 4: None, 5: None, 6: None, 7: 1, 8: 1}, {1: None, 2: 3, 3: 1, 4: 3, 5: 2, 6: 1, 7: 1, 8: 1},…]

#### LOCList

This key, returns a list of dictionaries that each dictionary shows the released liquid mass value (Loss Of Containment) of the plant unit in each scenario. Example:

Results['LOCList']

[{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0}, {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0}, {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0},{1: 2827433.3882308137, 2: 4208351.855042743, 3: 4208351.855042743, 4: 4208351.855042743, 5: 3674532.4313447657, 6: 3674532.4313447657, 7: 3674532.4313447657, 8: 3674532.4313447657},…]

#### NodesGroupDamageList

This key, return a list of dictionaries each dictionary is results of each scenario and its keys are the NodesGroups tag and corresponding value shows the elements damage condition according defined probit functions (0 shows No damage and 1 shows damaged). It returns empty list for Not damaged case. Example:

Results['NodesGroupDamageList']

[{1: []}, {1: []}, {1: []}, {1: []}, {1: []},{1:[0,0,0,0,0,1,1,1,0,1,0,0,0,1,1,1,1,0,0,0,1,0]}]

#### NodesGroupTypeDict

This key returns a dictionary that each key refers to a NodesGroup tag and the corresponding value is the type of the NodesGroup.

#### TotalLOCList

These key returns list of total liquid mass (kg) that has released in each scenario. Example:

Results['TotalLOCList']

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3674532.4313447657, 0, 0, 03674532.4313447657, 20593020.94647379, 0, 0, 0, 11557416.717732275, 5674351.416812722, 0, 0, 0, 14028296.729529412, 4208351.855042743, 0, 0, 0, 0, 0,…]

#### LOC\_bins\_hist\_probloc

This key returns 3 lists that are histogram data of the released liquids however they can be calculated from the previous described list. The first list is the bins data that its length should be one value greater than the two other lists. The second list is histogram data that shows the frequency of the bins and the last list is the probability of each bin value. Example:

[bins,hist,probloc]= Results['LOC\_bins\_hist\_probloc']

#### Total\_Number\_Of\_Scenarios

This key simply returns total number of sampled scenarios. Example

Results['Total\_Number\_Of\_Scenarios']

105000

#### UnitsZeroDamageProb

This key returns the damage probability of each unit in zero level. Example:

Results['UnitsZeroDamageProb']

{1: 0.0018952380952380952, 2: 0.001990476190476190, 3: 0.0021714285714285715, 4: 0.0019047619047619048, 5: 0.0019904761904761905, 6: 0.0021714285714285715, 7: 0.0021523809523809525, 8: 0.001961904761904762}

The keys are the units tag and their corresponding values show the probability of damaging in the zero level.

#### ProbOfFragilities

This key returns the probability of happening of each defined fragility or probit function. Example:

Results['ProbOfFragilities']

{1: 0.003952380952380952, 2: 0.012285714285714285, 3: 0.005238095238095238, 4: 0.0009904761904761905, 5: 0.0, 6: 0.0}

The keys show the defined fragility or probit tag and the corresponding values shows their governing probability among analysis. Probits that have defined for the Vulnerable areas (NodesGroup) will not consider in this part and their probability will be shown as zero (probits with tag 5 and 6).

#### Damagelevel\_eLOC

This key returns the expected released liquid (mass in kg) in each damage level. Example:

Results['Damagelevel\_eLOC']

{0: 56082.8170381438, 1: 8802.563549482884, 3: 925.6543716581589, 4: 395.0027138190845, 2: 3376.962598791185, 6: 26.927937030769655, 5: 116.56262508340092}

Each key refers to the damage level and the corresponding value shows expected liquid released mass in that damage level.

#### ScenariosAnalyzeNumbers

This key returns scenarios name with the following format as key and list of the analyze number as corresponding value. Example:

Results['ScenariosAnalyzeNumbers']

{'(0):[3]': [202, 5646, 16833, 17173, 20846, 23658, 29179, 30415, 41698, 42064, 42114], '(0):[3,5]-(1):[2]-(2):[1,6,7]': [316778], '(0):[1,3,6]-(1):[4,7]': [316830], '(0):[1,3,6]-(1):[4,7]-(2):[8]': [316830], '(0):[4,5]-(1):[3]-(2):[2]-(3):[1,6]': [316858]}

The rule of mentioning scenarios is : (Damage level):[list of units tag that damaged in this level] for example:

'(0):[3]' shows a scenario with damaging plant unit 3 in damage level 0

'(0):[3,5]-(1):[2]' shows a scenario with damaged plant units with tag 3 and 5 in damage level 0 and damaged plant unit with tag 2 at damage level 1

#### ScenariosProbability

This key returns scenarios name as key and the corresponding probability as value. Example:

Results['ScenariosProbability']

{'(0):[3]': 0.00026476190476190474, '(0):[3]-(1):[2]': 7.619047619047618e-05, '(0):[3]-(1):[2]-(2):[1]': 1.0476190476190477e-05, '(0):[1,2,3,4,5,6,7,8]': 0.000659047619047619, '(0):[8]': 0.00026285714285714286, '(0):[2,7,8]': 8.571428571428571e-06, '(0):[2,7,8]-(1):[3,6]': 2.8571428571428573e-06, …}

#### ScanariosSubScenario

This key returns a dictionary that its key is the Scenario name and the corresponding value is next damage level scenarios. Example:

Results['ScanariosSubScenario']['(0):[3]']

['(0):[3]-(1):[2]', '(0):[3]-(1):[4]', '(0):[3]-(1):[2,4]', '(0):[3]-(1):[7,8]', '(0):[3]-(1):[7]', '(0):[3]-(1):[2,6]', '(0):[3]-(1):[4,7]', '(0):[3]-(1):[4,7,8]', '(0):[3]-(1):[2,7]', '(0):[3]-(1):[1,2]', '(0):[3]-(1):[2,4,7]', '(0):[3]-(1):[5,6,7]', '(0):[3]-(1):[6,7]']

To see next level scenario:

Results['ScanariosSubScenario']['(0):[3]-(1):[2]']

['(0):[3]-(1):[2]-(2):[1]', '(0):[3]-(1):[2]-(2):[5,6]', '(0):[3]-(1):[2]-(2):[6]', '(0):[3]-(1):[2]-(2):[7]', '(0):[3]-(1):[2]-(2):[1,6]', '(0):[3]-(1):[2]-(2):[4]', '(0):[3]-(1):[2]-(2):[4,7]', '(0):[3]-(1):[2]-(2):[6,7]']

#### Damagelevel\_Scenario\_Dict

This key returns a dictionary that its keys are the damage level and its values are list of the Scenarios in the corresponding level. Example:

Results['*Damagelevel\_Scenario\_Dict*']

#### HazardMagnitude

This key returns a list that each cell is a dictionary that its key is the hazard tag and each value is the sampled value. Example:

Results['HazardMagnitude']

#### NodesGroupRadiationDict

This key returns a dictionary that its keys are the NodesGroup tag and the corresponding value is a list of each node radiation average values. Example:

Results['NodesGroupRadiationDict']

#### NodesGroupOverPressureDict

This key returns a dictionary that its keys are the NodesGroup tag and the corresponding value is a list of each node Overpressure average values. Example:

Results['NodesGroupOverPressureDict']

#### NodesGroup\_OVP\_Probit\_Dict

This key returns a dictionary that its keys are the NodesGroup tag and the corresponding value is a list of each node Overpressure probit average values [Probit(OverPressure)]. Example:

Results['*NodesGroup\_OVP\_Probit\_Dict*']

#### NodesGroup\_Rad\_Probit\_Dict

This key returns a dictionary that its keys are the NodesGroup tag and the corresponding value is a list of each node Radiation probit average values [Probit(Radiation)]. Example:

Results['*NodesGroup\_Rad\_Probit\_Dict*']

### RecorderPP

This module, gets the recorder files and analyze their data and returns a dictionary with the same results of ObjsRecorderPP. An example of the usage has been shown in the following:

Results=opr.PostProcess.RecorderPP.Analyze(Recorer\_FilenamesList=

['RecorderA.OPRrec','RecorderB.OPRrec','RecorderC.OPRrec','RecorderD.OPRrec','RecorderE.OPRrec'])

Above analyze returns a dictionary that its keys and values are same as described for ObjsRecorderPP.

### PlotPP

This module has various plot options to plot the obtained results of the analysis. It **gets** the **results of the** RecorderPP or ObjRecorderPP **as input variable** and it plots the following data:

#### DamageLevel\_ExpectedLoss

Plots the expected loss of containment in each damage level.

#### Unit\_ZeroLevel\_DamageProb

Plots each plant unit damage probability in zero level.

#### Fragilities\_Probits\_Probability

Plots each fragility and probit happening probability.

#### Expected\_Total\_LOC

Plots expected total loss of containment.

#### ScenarioProbability

Plots probability of scenarios.

## Plot Subpackage

Plot subpackage present some commands that using them user can plot some of defined or resulted data. Obviously, the current plots are according current needs and users can add other plots with other visualization packages and …. Plotly python visualization package has been selected for plots however users can develop other plotting packages. In the following, available plotting commands are described:

### Plotly.PlotFragilities

To plot defined fragilities. Example:

opr.Plot**.**Plotly**.**PlotFragilities**(**

StdNumber**=3,**

NPoints**=100,**

FragilityTagList**=[],)**

StdNumber is the number of the standard deviation that will be consider for each fragility to plot in this range. NPoints is the number of the points that will be consider for each fragility. FragilityTagList is the list of fragility tags and those that mentioned in this list will be plotted and if nothing entered all fragilities will be plot.

### Plotly.PlotProbits

To plot defined probits. Example:

opr.Plot**.**Plotly**.**PlotProbits**(**

StdNumber**=3,**

NPoints**=100,**

ProbitTag**=None)**

StdNumber is the number of the standard deviation that will be consider for each fragility to plot in this range. NPoints is the number of the points that will be consider for each fragility. ProbitTag is the tag of a special defined probit to plot and in nothing enter all probits will be plotted.

### Plotly.PlotHazard

To plot defined hazards. Example:

opr.Plot**.**Plotly**.**PlotHazard**()**

### Plotly.PlotWindRose

To plot defined windrose. Example:

opr.Plot**.**Plotly**.**PlotWindRose**(**WindRoseTag**,** Draw\_For\_Day**=True)**

### Plotly.PlotUnits2D

To plot the current plant in the memory. Example:

opr.Plot**.**Plotly**.**PlotUnits2D**(**

PlotMode**=1,**

OverPressureList**=[],**

OverPressureHeight**=2,**

OverPressurePointNumber**=20,**

RadiationList**=[],**

RadiationHeight**=2,**

RadiationPointNumber**=20,**

GasConcentrationlist**=[],**

GasConcentrationHeght**=2,**

ConcentrationPointNumber**=10,**

**)**

OverPressureList is list of overpressure values that user wants to be shown on plot and they will be calculated in OverPressureHeight height with OverPressurePointNumber points in boundary.

RadiationList is list of radiation values that user wants to be shown on plot and they will be calculated in RadiationHeight height with RadiationPointNumber points in boundary.

GasConcentrationlist is list of gas concentration values that user wants to be shown on plot and they will be calculated in GasConcentrationHeght height with ConcentrationPointNumber points in boundary.

### Plotly.PlotIndividualRisk

To plot individual risk. Example:

opr.Plot**.**Plotly**.**PlotIndividualRisk**(**

PlotMode**=1,**

NodesGroupTag**=1,**

NodesProbabilityList**=[],**

ContorList**=[],**

**)**

NodesGroupTag is the tag of NodesGroup that user want to be shown on plot. NodesProbabilityList is the probability values that has been resulted from analysis and the specify the death probability of each node (Obviously its length should be equal to the number of the NodesGroupTag nodes. ContorList is the list of the minimum and maximum contour value that we want to be plot.

# References

[1] J. Casal, *Evaluation of the Effects and Consequences of Major Accidents in Industrial Plants*, vol. 8. 2018.