Risk Assessment System Documentation

Contents

1 __init__ Documentation

1.1 Overview

This `__init__.py` file defines the public interface for a Python package related to risk assessment and hazard modeling. It imports and exposes various classes from different modules within the package.

1.2 Imported Classes

1.2.1 From `.assets` module:

1. `Asset`: Likely represents a general asset in the risk assessment context.
2. `PowerGeneratingAsset`: A specialized asset class, probably representing power plants or similar facilities.

1.2.2 From `.curve` module:

3. `ExceedanceCurve`: Likely used for probability calculations in risk assessment.

1.2.3 From `.hazard_event_distrib` module:

4. `HazardEventDistrib`: Probably represents the distribution of hazard events.

1.2.4 From `.hazards` module:

5. `Hazard`: A base class for different types of hazards.
6. `Drought`: A specific type of hazard representing drought conditions.
7. `RiverineInundation`: Another specific hazard type, likely representing flooding from rivers.

1.2.5 From `.vulnerability_distrib` module:

8. `VulnerabilityDistrib`: Likely represents the distribution of vulnerability for assets.

1.2.6 From `.vulnerability_model` module:

9. `VulnerabilityModelAcuteBase`: Probably a base class for acute vulnerability models.

1.3 Usage

When users import this package, they will have direct access to all these classes. For example:

```
from your_package_name import Asset, PowerGeneratingAsset, ExceedanceCurve, Haza

# Now you can use these classes directly
asset = Asset()
power_plant = PowerGeneratingAsset()
drought_hazard = Drought()
```

1.4 Package Structure

This `__init__.py` file suggests that the package has at least the following structure:

```
your_package_name/
|-- __init__.py
|-- assets.py
|-- curve.py
|-- hazard_event_distrib.py
|-- hazards.py
|-- vulnerability_distrib.py
`-- vulnerability_model.py
```

Each of these `.py` files contains the respective classes that are being imported in the `__init__.py` file.

1.5 Notes for Developers

- When adding new modules or classes to the package, remember to update this `__init__.py` file to expose them at the package level if needed.
- Be cautious about circular imports. The current structure seems to avoid this, but it's something to keep in mind as the package grows.
- Consider adding type hints and docstrings to these classes in their respective modules to improve IDE support and documentation.

2 Assets Module Documentation

2.1 Overview

This module defines various asset classes used in a risk assessment or asset management system, with a particular focus on power generating assets. It includes enumerations for fuel types, cooling systems, and turbine types, as well as classes for different types of assets.

2.2 Enumerations

2.2.1 `FuelKind` (Enum)

Represents different types of fuel used in power plants, based on the Global Power Plant Database v1.3.0.

Values include: Biomass, Coal, Cogeneration, Gas, Geothermal, Hydro, Nuclear, Oil, Other, Petcoke, Solar, Storage, Waste, WaveAndTidal, Wind

2.2.2 `CoolingKind` (Enum)

Represents different cooling systems used in power plants.

Values:

- `Dry`: Affected by Air Temperature, Inundation
- `OnceThrough`: Affected by Drought, Inundation, Water Temperature, Water Stress
- `Recirculating`: Affected by Drought, Inundation, Water Temperature, Water Stress, Wet-Bulb Temperature

2.2.3 `TurbineKind` (Enum)

Represents types of turbines used in power plants.

Values: Gas, Steam

2.3 Classes

2.3.1 `Asset`

Base class for all assets.

Attributes:

- `latitude` (float): Geographical latitude of the asset
- `longitude` (float): Geographical longitude of the asset
- `id` (Optional[str]): Unique identifier for the asset

2.3.2 `WindTurbine` (dataclass)

Represents a wind turbine, inheriting from `Asset`.

Additional attributes:

- `capacity` (Optional[float]): Power generation capacity
- `hub_height` (Optional[float]): Height of the turbine hub
- `cut_in_speed` (Optional[float]): Minimum wind speed for operation
- `cut_out_speed` (Optional[float]): Maximum wind speed for operation
- `fixed_base` (Optional[bool]): Whether the turbine has a fixed base (default: True)
- `rotor_diameter` (Optional[float]): Diameter of the rotor

2.3.3 `PowerGeneratingAsset`

Represents a generic power generating asset, inheriting from `Asset`.

Additional attributes:

- `type` (Optional[str]): Type of the power generating asset
- `location` (Optional[str]): Location of the asset
- `capacity` (Optional[float]): Power generation capacity
- `primary_fuel` (Optional[FuelKind]): Primary fuel used by the asset

2.3.4 `ThermalPowerGeneratingAsset`

Represents a thermal power generating asset, inheriting from `PowerGeneratingAsset`.

Additional attributes:

- `turbine` (Optional[TurbineKind]): Type of turbine used
- `cooling` (Optional[CoolingKind]): Type of cooling system used

Methods:

- `get_inundation_protection_return_period()`: Returns the design return period for inundation protection (250 years for most plants, 10,000 years for nuclear plants)

2.3.5 `RealEstateAsset`

Represents a real estate asset, inheriting from `Asset`.

Additional attributes:

- `location` (str): Location of the real estate
- `type` (str): Type of real estate

2.3.6 `ManufacturingAsset`

Represents a manufacturing asset, inheriting from `Asset`.

Additional attributes:

- `location` (Optional[str]): Location of the manufacturing asset
- `type` (Optional[str]): Type of manufacturing asset

2.3.7 `IndustrialActivity`

Represents an industrial activity, inheriting from `Asset`.

Additional attributes:

- `location` (Optional[str]): Location of the industrial activity
- `type` (str): Type of industrial activity

2.3.8 `TestAsset`

A simple test asset class, inheriting from `Asset` with no additional attributes or methods.

2.4 Usage Notes

1. The `Asset` class serves as the base for all other asset types, providing common attributes like latitude, longitude, and ID.
2. The `PowerGeneratingAsset` class uses a type string to determine the primary fuel. The type string can contain multiple archetypes separated by "/".
3. The `ThermalPowerGeneratingAsset` class extends this concept, using additional archetypes in the type string to determine turbine and cooling types.
4. The `get_inundation_protection_return_period()` method in `ThermalPowerGeneratingAsset` provides different protection levels for nuclear vs. non-nuclear plants.
5. Various asset types (RealEstate, Manufacturing, IndustrialActivity) are provided for different use cases in the risk assessment or asset management system.
6. The `TestAsset` class can be used for testing purposes or as a placeholder for future asset types.

3 Calculation Module Documentation

3.1 Overview

This module serves as a configuration hub for the physical risk assessment system. It defines default hazard models, vulnerability models, and risk measure calculators for various asset types. The module also includes a factory class for creating risk measure calculators based on specific use cases.

3.2 Imports

The module imports various components from the `physrisk` package, including hazard models, vulnerability models, and risk calculators. It also imports asset types from a local `.assets` module.

3.3 Functions

3.3.1 `get_default_hazard_model() -> HazardModel`

Returns the default hazard model, which is a `ZarrHazardModel` that retrieves hazard event data from Zarr storage.

3.3.2 `get_default_vulnerability_models() -> Dict[type, Sequence[VulnerabilityModelBase]]`

Returns a dictionary mapping asset types to sequences of vulnerability models. This function defines the default vulnerability models for different asset types:

- `Asset`: Generic asset models (placeholder for unknown assets)
- `PowerGeneratingAsset`: Inundation model
- `RealEstateAsset`: Models for coastal and riverine inundation, tropical cyclones, and cooling
- `IndustrialActivity`: Chronic heat model
- `ThermalPowerGeneratingAsset`: Models for various thermal power plant vulnerabilities
- `TestAsset`: Temperature model

3.3.3 `get_default_risk_measure_calculators() -> Dict[Type[Asset], RiskMeasureCalculator]`

Returns a dictionary mapping asset types to risk measure calculators. Currently, it only defines a calculator for `RealEstateAsset`.

3.4 Classes

3.4.1 `DefaultMeasuresFactory(RiskMeasuresFactory)`

A factory class for creating risk measure calculators based on specific use cases.
Methods:

- `calculators(self, use_case_id: str) -> Dict[Type[Asset], RiskMeasureCalculator]`:
  - If `use_case_id` is "generic", returns a dictionary with a `GenericScoreBasedRiskMeasures` calculator for the `Asset` type.
  - Otherwise, returns the result of `get_default_risk_measure_calculators()`.

3.5 Key Components

1. **Hazard Model**: The default hazard model uses Zarr storage for retrieving hazard event data.
2. **Vulnerability Models**:
   - For generic assets, placeholder models are used for various hazards (fire, chronic heat, hail, drought, precipitation).
   - Specific models are defined for power generating assets, real estate assets, industrial activities, and thermal power generating assets.
   - These models cover a range of hazards including inundation, tropical cyclones, chronic heat, and various thermal power plant-specific vulnerabilities.
3. **Risk Measure Calculators**:
   - The default setup includes a specific calculator for real estate assets (`RealEstateToyRiskMeasures`).
   - The `DefaultMeasuresFactory` allows for dynamic selection of risk measure calculators based on the use case.

3.6 Usage Notes

1. This module serves as a central configuration point for the physical risk assessment system. Users can modify these functions to customize the models and calculators used in their assessments.
2. The `get_default_vulnerability_models()` function allows for easy extension to support new asset types or vulnerability models.
3. The `DefaultMeasuresFactory` class provides a flexible way to select risk measure calculators based on different use cases. Users can extend this class to support additional use cases.

4. When adding new asset types or models, ensure they are imported correctly and added to the appropriate dictionaries in this module.

5. The use of placeholder vulnerability models (e.g., `PlaceholderVulnerabilityModel`) suggests that some parts of the system may still be under development or pending more specific implementations.