ExposureMeasure (ABC) Detailed Explanation

1 Overview

`ExposureMeasure` is an abstract base class (ABC) that defines the interface for exposure measure implementations. It serves as a blueprint for creating different types of exposure measures in the system.

2 Class Definition

```
class ExposureMeasure(DataRequester):
    @abstractmethod
    def get_exposures(
        self, asset: Asset, data_responses: Iterable[
    HazardDataResponse]
    ) -> Dict[type, Tuple[Category, float, str]]: ...
```

3 Key Aspects

1. **Inheritance from DataRequester**: `ExposureMeasure` inherits from `DataRequester`, which likely defines methods for requesting data. This suggests that exposure measures are responsible for specifying what data they need.

2. **Abstract Method**: The class defines one abstract method, `get_exposures`, which must be implemented by any concrete subclass.

3. **Method Signature**:

   - `asset: Asset`: The asset for which exposure is being calculated.
   - `data_responses: Iterable[HazardDataResponse]`: Responses from hazard data requests.
   - Returns `Dict[type, Tuple[Category, float, str]]`: A mapping of hazard types to exposure results.

4. **Return Value Structure**:

   - `type`: Likely represents the hazard type (e.g., Flood, Fire).

- `Category`: An enum representing the exposure category (e.g., LOW, MEDIUM, HIGH).
- `float`: The numerical value of the exposure.
- `str`: Probably a path or identifier for the data source.

4 Purpose and Design Philosophy

1. **Separation of Concerns**: By defining an abstract base class, the module separates the interface of exposure measures from their implementations. This allows for different exposure calculation methodologies without changing the overall system structure.
2. **Polymorphism**: The abstract class enables polymorphic use of different exposure measures. Functions like `calculate_exposures` can work with any concrete implementation of `ExposureMeasure`.
3. **Extensibility**: New exposure measures can be easily added by creating new classes that inherit from `ExposureMeasure` and implement the `get_exposures` method.
4. **Standardization**: The ABC ensures that all exposure measures have a consistent interface, making the system more maintainable and easier to understand.

5 Usage and Implementation

To create a new exposure measure:

1. Create a new class that inherits from `ExposureMeasure`.
2. Implement the `get_exposures` method, ensuring it adheres to the defined signature.
3. Optionally, implement any methods from `DataRequester` if needed.

Example skeleton:

```python
class NewExposureMeasure(ExposureMeasure):
    def get_exposures(self, asset: Asset, data_responses: Iterable[HazardDataResponse]) -> Dict[type, Tuple[Category, float, str]]:
        # Implementation here
        pass

    # Optionally implement DataRequester methods
    def get_data_requests(self, asset: Asset, *, scenario: str, year: int) -> Iterable[HazardDataRequest]:
        # Implementation here
        pass
```

6 Considerations for Implementers

1. **Data Handling**: Implementations need to handle various types of `HazardDataResponse` objects correctly.
2. **Categorization Logic**: Define clear logic for categorizing exposure levels based on the data.
3. **Error Handling**: Consider how to handle missing or invalid data in the responses.
4. **Performance**: For large-scale applications, consider the efficiency of the implementation, especially if processing many assets.