AssetExposureResult Detailed Explanation

1 Class Definition

```python
@dataclass
class AssetExposureResult:
    hazard_categories: Dict[type, Tuple[Category, float, str]]
```

2 Overview

`AssetExposureResult` is a dataclass that encapsulates the exposure results for a single asset across multiple hazard types. It provides a structured way to store and access the categorized exposure levels, numerical values, and data sources for each hazard type applicable to an asset.

3 Components

3.1 `@dataclass` Decorator

- This decorator automatically generates several methods for the class, including `__init__`, `__repr__`, and `__eq__`.
- It simplifies the class definition by allowing you to focus on declaring the fields.

3.2 `hazard_categories` Field

- Type: `Dict[type, Tuple[Category, float, str]]`
- This is the sole field of the class, containing all the exposure information for an asset.

3.2.1 Key (Dict Key): `type`

- Represents the hazard type (e.g., `Wind`, `Fire`, `ChronicHeat`).
- Using the type as a key allows for easy and type-safe access to specific hazard results.

3.2.2 Value (Dict Value): `Tuple[Category, float, str]`

1. `Category`: An enum representing the exposure category (e.g., `LOW`, `MEDIUM`, `HIGH`).
2. `float`: The numerical value of the exposure measure.
3. `str`: A string representing the data source or path.

4 Usage

```python
# Creating an instance
result = AssetExposureResult(hazard_categories={
    Wind: (Category.MEDIUM, 95.5, "wind/jupiter/v1/max_1min_RCP8.5_2050"),
    ChronicHeat: (Category.HIGH, 25.3, "chronic_heat/days_above_35c"),
    Fire: (Category.LOW, 0.15, "fire/probability")
})

# Accessing results
wind_category, wind_value, wind_source = result.hazard_categories[Wind]
print(f"Wind exposure: {wind_category.name}, Value: {wind_value}, Source: {wind_source}")

# Iterating over all hazards
for hazard_type, (category, value, source) in result.hazard_categories.items():
    print(f"{hazard_type.__name__}: {category.name} (Value: {value}, Source: {source})")
```

5 Key Features

1. **Typed Structure**: The use of types as dictionary keys provides clear and type-safe access to hazard-specific results.
2. **Comprehensive Information**: Each hazard entry contains not just the category, but also the numerical value and data source, providing context for the categorization.
3. **Flexibility**: Can accommodate any number of hazard types, allowing for easy extension to new hazards.
4. **Immutability**: As a dataclass, it's implicitly frozen (immutable) unless specified otherwise, ensuring data integrity after creation.
5. **Easy Serialization**: The simple structure makes it straightforward to serialize and deserialize, useful for storage or transmission.

6 Considerations and Potential Improvements

1. **Validation**: Consider adding validation to ensure all required hazard types are present and values are within expected ranges.

2. **Methods**: You might want to add methods for common operations, like getting all high-risk hazards or calculating an overall risk score.

3. **Customization**: If needed, you could add a `__post_init__` method to perform any necessary post-initialization processing or validation.

4. **Documentation**: Adding docstrings would improve usability, especially if this class is part of a larger API.

5. **Type Hinting**: Consider using more specific types (e.g., `Type[Hazard]` instead of `type`) for better type checking and IDE support.