OperationStatus Enum

Namespace: `Heatington.Controllers.Enums`

Description

Enumeration representing the status of an operation.

Members

- `SUCCESS` = 0: Indicates a successful operation.
- `LOADING` = 1: Indicates that the operation is in progress or loading.
- `FAILURE` = 2: Indicates a failed operation.

`IDataSource` Interface

Overview

The `IDataSource` interface serves as a protocol for managing data sources in the Heatington application. It stipulates the provision of methods to perform read and write operations on data.

Methods

`Task<List<DataPoint>?> GetDataAsync(string filePath)`

The `GetDataAsync` method signifies an asynchronous operation for retrieving data from a specified data source. The input `string filePath` represents the path to the file containing the pertinent data.

The method is expected to return a `List<DataPoint>` object, which contains the data of interest - specifically the heat demand and electricity price details. In cases where the data retrieval is unsuccessful or if there is no data present at the given file location, the method may return `null`.

`void SaveData(List<DataPoint> data, string filePath)`

The `SaveData` method is purposed towards storing `DataPoint` objects into a CSV file located at a specific file path. The `List<DataPoint> data` argument contains the data points that are set to be saved. The `string filePath` is an argument that provides the location at which the CSV file will be written to or overwritten.

Considering that its return type is `void`, all complications that arise during the data-saving process should be conveyed via exceptions.

This method is currently not implemented

Implementations

Any classes that function as Csv data sources within the Heatington application should implement this interface. This allows for consistency in the management of data across varying data sources and enables smoother transitions between different data sources. Examples of such classes could include `CsvDataSource`, `XmlDataSource`, and the like.

IReadWriteController Interface

Namespace: `Heatington.Controllers.Interfaces`

Description

Generic Interface for all controllers performing I/O operations.

Members

`Task<T?> ReadData<T>()`

Function for reading data out of a model.

Returns

Task to wait for Data.

`Task<OperationStatus> WriteData<T>(T content)`

Function for Writing data into the model.

Parameters

- `content`: Content to write into the model, generic type.

Returns

Task to wait for status of the operation.

NOT IMPLEMENTED

`CsvController` Class

Overview

The `CsvController` class provides a concrete implementation of the `IDataSource` interface specific to data in CSV format, allowing for the reading of such data within the Heatington application.

Methods

`Task<List<DataPoint>?> GetDataAsync(string filePath)`

The `GetDataAsync` method is a function for asynchronously fetching data from a CSV file located at a provided file path.

The method initiates by asynchronously reading the file's entire content into a string (`rawData`). The `rawData` is then deserialized by the `CsvController` utility class into a `CsvData` object – a controller specifically designed to handle and manipulate data in CSV format.

Following the successful deserialization of the `rawData`, the `CsvData` object is converted into a `List` of `DataPoint` objects. Each `DataPoint` object encapsulates heat demand and electricity price data.

Should the method fail to retrieve data from the file or if no data exists at the provided file path, then the method will return `null`.

Upon the occurrence of an exception, the exception's message is displayed using the `Utilities.DisplayException(e.Message)` method and the exception is then re-thrown.

`void SaveData(List<DataPoint> data, string filePath)`

The `SaveData` method remains unimplemented and consequently triggers a `NotImplementedException` when called.

It is projected that in the future, the method will save a `List` of `DataPoint` objects into a CSV file located at a specified file path. The `List<DataPoint> data` parameter represents the data points to be stored. The `string filePath` parameter provides the location of where the CSV file will be created or rewritten.

The utilization of this method remains dependent upon the needs of the Heatington project.

Remarks

While the `CsvController` class is intended to offer a concrete manner for handling CSV data in the application, it's worth to mention that its ability to write data is not implemented. Depending on future development decisions, this feature could potentially remain so. The future of object creation and modification may also leverage design patterns such as the Factory or Builder.

FileController Class

Namespace: `Heatington.Controllers`

Example

```
string pathToFile = Utilities.GeneratePathToFileInAssetsDirectory("testFile.json");
IReadWriteController fileController = new FileController(pathToFile);
await fileController.WriteData("1");
string? data = await fileController.ReadData();
```

Description

Class for performing read/write actions on local files.

Constructor

`FileController(string pathToFile)`

Class constructor. Gets a path to the file to control.

Parameters

- `pathToFile` (string): Path to the location of a file. Follows the pattern `file.json`.

Members

`TryFileOperationRunner<T>(Func<Task<T>> funcToTry)`

Helper method performing try-catch clauses in file-oriented manner.

Parameters

- `funcToTry` (Func<Task>): Function to run inside of try-catch block.

Type Parameters

- `T`: Return type of a function.

Returns

Return the outcome of the run function. If the function returns void, the lambda function should return 0.

`ReadFileFromPath()`

Function for reading the contents of a local file. Reads from the path passed during object initialization.

Returns

Content of the file as a string.

Exceptions

- `FileNotFoundException`: If the file does not exist or the path is not correct, the exception is thrown.

`WriteToFileFromPath(string content)`

Function for writing string data into a local file from a path.

Parameters

- `content` (string): Content to be written to a file as a string.

`ReadData<T>()`

Function for reading the data of a local file. Reads from the path passed during object initialization.

Returns

Data of the file as a string.

`WriteData<T>(T content)`

Function for writing data into a local file.

Parameters

- `content` (T): Content to be written to a file as a string.