



*Azure Function Code Explanation*

*For*

FOCUS-BI – Cylindrical Oil Feed Rate Report (File Processing)

*Author: MOL-IT AID*

*Date Created: 22-Nov-2023*

*Last Revision:*

*Revised By:*

**Guidelines**

**Table of Contents**

[*Overview:* 3](#_Toc151548704)

[*Folder Name: CO\_File\_Processing\_Act* 4](#_Toc151548705)

[*File Name: CO\_Process.py* 7](#_Toc151548706)

[*File Name: az\_datalake\_activity.py* 10](#_Toc151548707)

# Overview:

This documentation provides an overview of each file's role within the Azure Durable Functions application, from activity processing to orchestrator coordination and blob-trigger-based execution.

Our system utilizes Azure Blob Storage to manage incoming data files. These files are subject to a structured process triggered by a blob update event, invoking a specific Azure Function. This Azure Function is designed to process the received files, make necessary updates, and then replace them in the storage.

The workflow can be outlined as follows:

* Trigger: A blob-triggered Azure Function is initiated upon the update of a designated file, referred to as the "CO\_Incremental\_Data.csv". It is placed on “nonprdfocusbisto” container in a folder named “ABLOG\_FINAL/Cylindrical\_Oil/Incremental\_Data”. This file will automatically be updated daily by a different azure function “CO\_File\_Fetching”.
* Processing: The Azure Function is responsible for processing the files, performing essential operations, and subsequently updating other files as required.
* Archiving: After successful processing, the originally received files are systematically copied to an archive folder for historical record-keeping.
* Deletion: To maintain an organized storage environment, the processed files are removed from their original location in the source directory.
* Logging: In addition to file processing, log files are generated and uploaded to retain a comprehensive record of the function's execution and data manipulation.

This workflow ensures the efficient management of incoming data, accurate processing, and proper archival, while also maintaining a detailed log of all activities performed by the Azure Function.

Files Description:

1. Incremental\_Data:

* Purpose: This daily-updated file contains incremental data, essential for our processing tasks. Its content evolves daily, necessitating a daily archival process to preserve historical records.

1. Initial\_Data:
   * Purpose: This file contains the full data used in the dashboard. The data from the “Incremental\_Data” File will append into this file on daily basis.

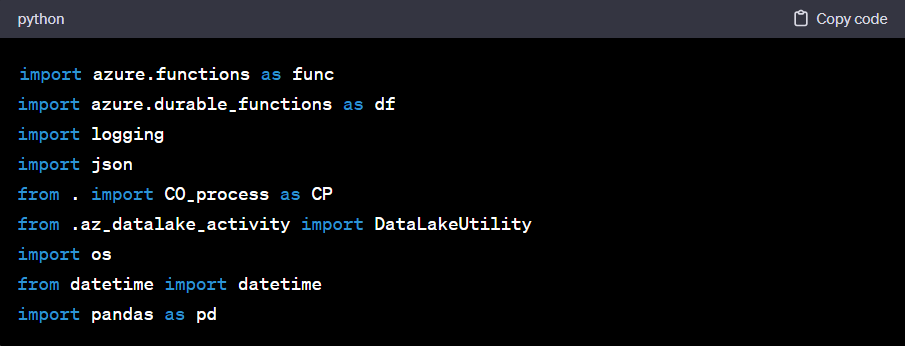
# Folder Name: CO\_File\_Processing\_Act

**File Name: \_\_init\_\_.py**

Description:

The provided Python code is an Azure Durable Function activity that orchestrates the download, processing, and upload of data to Azure Blob Storage. It utilizes a **DataLakeUtility** class for managing operations on Azure Data Lake Storage, and custom functions in the **CO\_process** module for processing CSV data using Pandas. The code includes logging, error handling, and a copy-and-delete operation on Azure Blob Storage. The overall purpose is to automate data processing workflows in a distributed and scalable environment.

1. Importing Libraries and Modules:



The code begins by importing necessary libraries and modules for working with Azure Durable Functions, data processing, and interacting with Azure services.

1. Initializing Variables:

A screen shot of a computer

Description automatically generated

Global variables are initialized, including an instance of **DataLakeUtility**, the current date, the blob path for logging, and the destination container name for uploading processed data.

1. Main Function – ‘main’:

A black screen with a black background

Description automatically generated with medium confidence

The **main** function is the entry point for the activity. It takes a **name** parameter, for personalization, and returns a string.

1. Logging Activity:

A computer screen with white text

Description automatically generated

A list **log\_Activity** is initialized to keep track of activity progress. Logging statements are used to document various steps within the activity function.

1. Exception Handling:

A screen shot of a computer

Description automatically generated

In case of any exceptions during the execution, the error message is logged, and the log data is uploaded to Azure Blob Storage. The **finally** block ensures that the log data is always uploaded, even in the presence of exceptions.

1. File Download:

A screen shot of a computer

Description automatically generated

Files are downloaded from Azure Blob Storage to a temporary folder using the **download\_blob** method of **DataLakeUtility**.

1. Processing CSV File with Pandas:

A screen shot of a computer

Description automatically generated

CSV files are processed using custom functions from **CO\_process** module. Pandas is used to perform operations like sorting, filtering, and concatenation.

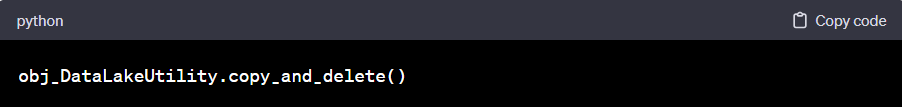
1. Uploading Processed Data:

A screen shot of a computer screen

Description automatically generated

The final processed data is saved to a temporary CSV file, and then it is uploaded to Azure Blob Storage under the specified container, path, and name.

1. Copy and Delete Operation:



The **copy\_and\_delete** method is called to perform a copy and delete operation on Azure Blob Storage, for managing data versions or archiving.

1. **Function Completion:**

****

If the function completes successfully, it returns a greeting message.

In summary, the provided code represents an Azure Durable Function activity that downloads, processes, and uploads data to Azure Blob Storage. It incorporates logging, error handling, and file operations for managing data in a distributed and scalable environment.

# File Name: CO\_Process.py

Description:

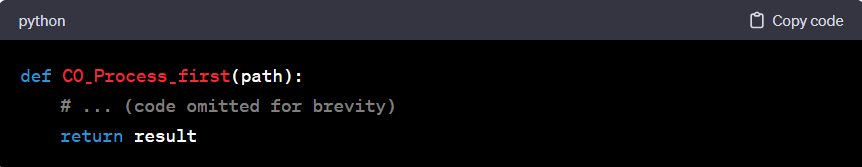
The code defines two functions for processing data. **CO\_Process\_first** handles initial processing, including column conversions, sorting, and conditional logic, while **CO\_Process\_Second** performs further processing, extracting hours, replacing values, and calculating cumulative propelling hours. The code emphasizes modularity and readability, making it suitable for data processing workflows.

1. Importing Libraries:



The code begins by importing the Pandas and NumPy libraries, commonly used for data manipulation and numerical operations, respectively.

1. ‘CO\_Process\_first’ Function:



This function processes a CSV file located at the specified **path** using Pandas. It includes steps such as converting columns to appropriate data types, sorting the DataFrame, calculating a new column ('PropHours'), and applying conditional logic.

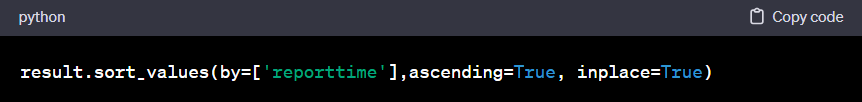
* Converting Columns to Appropriate Data Types:

A screen shot of a computer

Description automatically generated

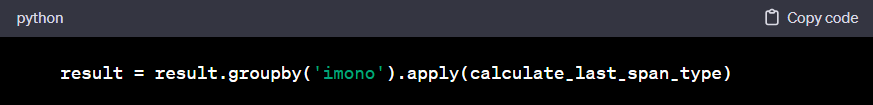
The 'reporttime' and 'spantime' columns are converted to datetime objects, handling any conversion errors with NaT (Not a Time) values.

* Sorting Dataframe:



The DataFrame is sorted by the 'reporttime' column in ascending order. Sorting is crucial for the subsequent logic to work correctly.

* Creating ‘LastSpanType’ Column:



A custom function (**calculate\_last\_span\_type**) is applied to create a new column 'LastSpanType,' representing the 'spantype' of the previous row for each 'imono' group.

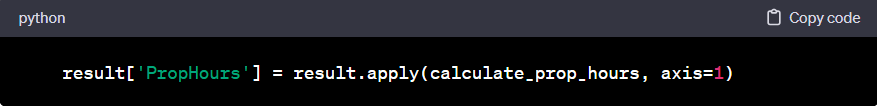
* Resetting Index and Handling Missing Values:

A screen shot of a computer code

Description automatically generated

Here, the index is reset after the groupby operation. Additionally, any 'NaT' values in the 'reporttime' and 'spantime' columns are replaced with empty strings for better consistency.

* + PropHours Logic:



A new column 'PropHours' is created by applying a custom function (**calculate\_prop\_hours**) to each row of the DataFrame. This column represents calculated hours based on specific conditions.

1. ‘CO\_Process\_Second’ Function:

A black screen with white text

Description automatically generated

This function processes the DataFrame produced by **CO\_Process\_first**. It includes steps such as extracting hours from the 'PropHours' column, replacing 'NaT' values with 0, and calculating cumulative propelling hours.

Custom Functions for Further Processing:

Several custom functions are defined for additional processing steps. Let's explain each of them:

* ‘extract\_hours’:

A black and white screen

Description automatically generated

This function extracts hours from a given value, handling different cases like NaN, NaT, or invalid values. It returns the extracted hours as a decimal.

* ‘replace\_NaT\_with\_0’:

A black and white screen

Description automatically generated

This function replaces 'NaT' values with 0, ensuring consistency in the 'PropHours' column.

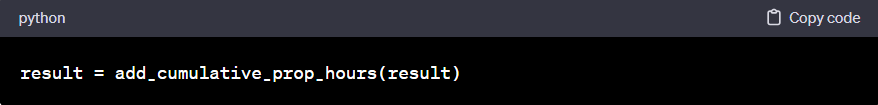
* ‘add\_cumulative\_prop\_hours’:

A screen shot of a computer

Description automatically generated

This function calculates cumulative propelling hours based on specific conditions, adds new columns ('NewPropHours' and 'CumuPropHours') to the DataFrame, and returns the updated DataFrame.

1. Applying ‘add\_cumulative\_prop\_hours’ Function:



This line applies the **add\_cumulative\_prop\_hours** function to the DataFrame, completing the overall data processing logic.

In summary, the **CO\_Process** function orchestrates a series of operations to preprocess and calculate propelling hours for the given DataFrame. Custom functions are used for specific tasks, promoting modularity and readability in the code. The result is a DataFrame with additional columns representing calculated and cumulative propelling hours.

# File Name: az\_datalake\_activity.py

Initialization (\_\_init\_\_): This function initializes the DataLakeUtility object. It sets attributes like guid (a unique identifier for the instance), root\_folder (the temporary directory where downloaded files will be stored), and connection\_string (the connection string to access Azure Blob Storage).

Function Name: connect\_azure

* This method establishes a connection to Azure Blob Storage using the connection string. It returns a BlobServiceClient object that allows you to interact with the Azure Blob service.

Function Name: download\_blob

* This function downloads a specific blob from Azure Blob Storage to a local temporary directory. It takes the container name (contr), blob path (blob\_path), and the file name (file\_name) to save the downloaded file in the local directory. It returns a dictionary with information about the download operation, including the local file's absolute path.

Function Name: upload\_blob

* This method uploads a local file from the temporary directory to Azure Blob Storage. It takes the container name (contr), blob path (blob\_path), the local file's absolute path (temp\_file\_abs\_path), and the desired blob name (blob\_name). It uploads the file and returns a dictionary with information about the upload operation.

Function Name: upload\_log

* This function is specifically designed for uploading log data to Azure Blob Storage. It takes the log data string, blob name, blob path, and the destination container name. It uploads the log data to the specified location and returns a message indicating a successful upload.

**Function Name: delete\_dir**

* This function deletes the temporary directory created for the instance. If the directory exists, it is removed. It returns a dictionary with information about the deletion, including the directory location.

Function Name: copy\_and\_delete

* This method is designed to copy files from one Azure Blob Storage container to another container and then delete them from the source container. It's used for archiving data. The function performs the following steps:
  + Initializes Azure Blob Service clients for both the source and destination containers.
  + Copies specified files from the source container to the destination folder and marks them for deletion.
  + Deletes the copied files from the source container.
  + Logs messages indicating the successful copy and deletion of files.

Please note that in functions like download\_blob and upload\_blob, error handling and logging are implemented to provide detailed information about the operations. The class provides an interface to interact with Azure Blob Storage for common data transfer and archival tasks, making it a useful utility for managing data in Azure Blob Storage.