using Azure.Storage.Blobs;

using CsvHelper;

using CsvHelper.Configuration;

using CsvHelper.Configuration.Attributes;

using Google.Protobuf.WellKnownTypes;

using PSVFunctionApp2;

using System.Collections.Generic;

using System.ComponentModel;

using System.Globalization;

using System.Net;

using System.Reflection.PortableExecutable;

using System.Text;

using System.Text.Json;

using System.Text.Json.Serialization;

namespace PSVFunctionApp

{

public class PSVProductionCalender

{

public byte[] AppendRecords(List<ProductionCalender> records, StringBuilder contentBuilder)

{

try

{

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = "|",

HeaderValidated = null,

MissingFieldFound = null

};

// Write modified data back to StringBuilder

contentBuilder.Clear();

using (var writer = new StringWriter(contentBuilder))

using (var csv = new CsvWriter(writer, config))

{

csv.WriteRecords(records);

}

// Convert StringBuilder to Stream

return Encoding.UTF8.GetBytes(contentBuilder.ToString());

}

catch (Exception ex)

{

throw ex;

}

}

public byte[] UpdateAddRecord(StringBuilder contentBuilder, ProductionCalender? newrecord)

{

byte[] bytearray = null;

bool newProductionCalender = true;

var config = new CsvConfiguration(CultureInfo.InvariantCulture)

{

Delimiter = "|",

HeaderValidated = null,

MissingFieldFound = null ,

};

List<ProductionCalender> records;

var recordlist = new List<ProductionCalender>();

try

{

using (var reader = new StringReader(contentBuilder.ToString()))

using (var csv = new CsvReader(reader, config))

{

records = csv.GetRecords<ProductionCalender>().ToList();

}

foreach (var item in records)

{

if (item.StartTime == newrecord.StartTime

&& item.EndTime == newrecord.EndTime && item.Date == newrecord.Date && item.RunType == newrecord.RunType && item.Shift == newrecord.Shift)

{

newProductionCalender = false;

item.Date = newrecord.Date;

item.Line = newrecord.Line;

item.MaxLinespeedTOC = newrecord.MaxLinespeedTOC;

item.Facility = newrecord.Facility;

item.Shift = newrecord.Shift;

item.LightsOnFlag = newrecord.LightsOnFlag;

item.LightsOffFlag = newrecord.LightsOffFlag;

item.MaxSpeedDueToBottleneckPackaging = newrecord.MaxSpeedDueToBottleneckPackaging;

item.EndTime = newrecord.EndTime;

item.StartTime = newrecord.StartTime;

item.PeriodWeek = newrecord.PeriodWeek;

item.TimeMins = newrecord.TimeMins;

// recordlist.Add(item);

}

recordlist.Add(item);

}

if (newProductionCalender && newrecord!=null) {

recordlist.Add(newrecord);

}

if (recordlist != null)

{

bytearray = AppendRecords(recordlist, contentBuilder);

}

return bytearray;

}

catch (Exception ex)

{

throw ex;

}

}

}

}

[Delimiter("|")]

[CultureInfo("en-US")]

public class ProductionCalender

{

// validFrom|facility|equipement|localName|Value stream|line|Area|Sub Area|Cell|Unit|S88\_Area

// |S88\_Process Cell|S88\_Unit|S88\_name|Machine Type|Bottleneck Flag|Provider Flag|

// Direct Consumer Flag

// |EndPoint Flag|TOC|Vendor|Version|ERP ID

// Date|StartTime|EndTime|Facility|Line|shift|RunType|MaxLinespeedTOC|MaxSpeedDueToBottleneckPackaging|LightsOnFlag|LightsOffFlag

public string? Date { get; set; }

public string? StartTime { get; set; }

public string? EndTime { get; set; }

public string? Facility { get; set; }

public string? Line { get; set; }

public string? Shift { get; set; }

public string? PeriodWeek { get; set; }

public string? RunType { get; set; }

public string? TimeMins { get; set; }

public string? MaxLinespeedTOC { get; set; }

public string? MaxSpeedDueToBottleneckPackaging { get; set; }

public string? LightsOnFlag { get; set; }

public string? LightsOffFlag { get; set; }

}

public class EnvProductionCalender

{

public ProductionCalender? productionCalender { get; set; }

public string? Environment { get; set; }

public EnvProductionCalender()

{

this.productionCalender = new ProductionCalender();

}

}

The code is part of an Azure Function App for managing production calendar data in CSV format. It involves operations to append, update, and manage records of production calendars. Here’s a detailed look at each component and its functionality:

**1. Namespace and Usings**

* **Namespaces and Usings**:
  + **Azure.Storage.Blobs**: Used for interacting with Azure Blob Storage (though not used directly in the provided code).
  + **CsvHelper**: A library for reading and writing CSV files in .NET.
  + **Google.Protobuf.WellKnownTypes**: This namespace includes Protobuf types but is not used in the provided code.
  + **System**: Contains basic classes and functions, such as for handling exceptions and working with text encodings.
  + **System.Text**: Provides classes for encoding and decoding text.
  + **System.Text.Json**: Provides classes for working with JSON (not used in the provided code but included).

**2. PSVProductionCalender Class**

This class handles operations related to production calendar data in CSV format.

**AppendRecords Method**

* **Purpose**: Converts a list of ProductionCalender objects into CSV format and returns it as a byte array.
* **Parameters**:
  + List<ProductionCalender> records: A list of ProductionCalender records to be written to CSV.
  + StringBuilder contentBuilder: A StringBuilder object that holds the CSV content.
* **Process**:
  + **Configuration**: Initializes CsvConfiguration with a pipe (|) as the delimiter and disables header validation and missing field detection.
  + **Write Records**:
    - Clears the contentBuilder to ensure it starts empty.
    - Uses CsvWriter to write the list of records into the contentBuilder.
  + **Convert to Byte Array**: Converts the contents of contentBuilder to a UTF-8 encoded byte array and returns it.

**UpdateAddRecord Method**

* **Purpose**: Updates an existing record in the CSV data or adds a new one if it doesn’t exist, and returns the updated data as a byte array.
* **Parameters**:
  + StringBuilder contentBuilder: Contains the current CSV data.
  + ProductionCalender? newrecord: The record to be updated or added.
* **Process**:
  + **Configuration**: Sets up CsvConfiguration similarly to the AppendRecords method.
  + **Read Records**:
    - Reads the current CSV data from contentBuilder into a list of ProductionCalender objects using CsvReader.
  + **Update/Add Logic**:
    - Iterates through each existing record and checks if it matches the newrecord based on several properties (StartTime, EndTime, Date, RunType, Shift).
    - If a match is found, updates the existing record with values from newrecord.
    - Adds the updated or new record to a recordlist.
  + **Write Updated Records**:
    - Calls AppendRecords to write the updated list of records back to CSV format.
  + **Return**: Converts the updated CSV data to a byte array and returns it.

**3. ProductionCalender Class**

* **Purpose**: Defines the structure of each production calendar record used in CSV.
* **Properties**:
  + **Date**: The date of the production.
  + **StartTime**: The start time of the production.
  + **EndTime**: The end time of the production.
  + **Facility**: The facility where production takes place.
  + **Line**: The production line.
  + **Shift**: The shift during which production occurs.
  + **PeriodWeek**: The period week for the production.
  + **RunType**: Type of production run.
  + **TimeMins**: Duration of production in minutes.
  + **MaxLinespeedTOC**: Maximum line speed.
  + **MaxSpeedDueToBottleneckPackaging**: Maximum speed due to bottlenecks in packaging.
  + **LightsOnFlag**: Flag indicating if the lights are on.
  + **LightsOffFlag**: Flag indicating if the lights are off.

**4. EnvProductionCalender Class**

* **Purpose**: Provides a wrapper for a ProductionCalender object along with an environment context.
* **Properties**:
  + **ProductionCalender? productionCalender**: Holds an instance of ProductionCalender.
  + **string? Environment**: Stores information about the environment in which the production calendar is used.
* **Constructor**:
  + Initializes the productionCalender property with a new instance of ProductionCalender.

**Summary**

* **AppendRecords**: Converts a list of production calendar records into a CSV format and returns it as a byte array.
* **UpdateAddRecord**: Reads CSV data, updates or adds records, and returns the updated CSV data as a byte array.
* **ProductionCalender**: Defines the schema for production calendar records.
* **EnvProductionCalender**: Wraps a production calendar record with additional environment context.

The code is designed to handle CSV data manipulation for production calendars, enabling updates and additions of records while working with CSV data in a flexible manner.