Digital Text Extraction and OCR

Comprehensive PDF Information Extractor and CSV Generator

1. Introduction

***Overview of the Project**:*

This project focuses on optimizing and enhancing an existing OCR software system, which is used to convert scanned documents into text and extract specific information such as issue dates, invoice numbers, and supplier details. The enhancement includes skipping unnecessary OCR processes for already digital documents, which improves the system's efficiency and accuracy.

***Purpose and Objectives**:*

The main objective was to improve the existing OCR system by introducing a function to handle digital documents without performing OCR, thereby avoiding potential errors in text extraction from already digitalized documents.

2. Existing System Overview

Description of the Old Software:

The previous system was capable of performing OCR on any document, extracting necessary details, and storing them in a journal for tracking purposes. It included a main function that managed the OCR and information extraction processes.

Identified Issues and Limitations:

A significant problem was that the system performed OCR on all documents, including those that were already digital (e.g., online invoices), which was unnecessary and could lead to inaccuracies when the text was extracted from images generated during the OCR process.

3. New Enhancements

SkipOCR Functionality:

The newly introduced function, `SkipOCR`, does not determine if a document is digital. Instead, a separate function, `Is-OCRed`, is responsible for checking whether a PDF document is already OCR-ed or not. If the document is found to be digital, the `SkipOCR` function is invoked to extract text directly using the `pdftotext` tool, ensuring that the extracted information is accurate and free of OCR-induced errors.

Improvements in Efficiency and Accuracy:

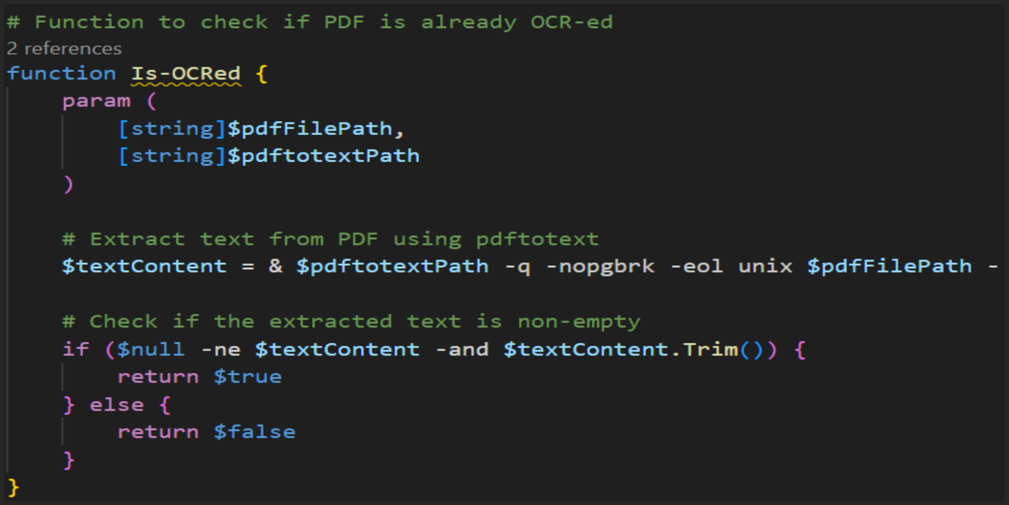
By checking if a document is already OCR-ed and skipping the OCR process when it is not needed, the new method significantly reduces processing time and eliminates errors introduced during the OCR of already digital documents.

4. Technical Implementation

Main Function Workflow:

Is-OCRed Function:

This function checks if the PDF document is already OCR-ed by using the `pdftotext` tool to extract text from the document. If the extracted text is non-empty, the document is deemed already OCR-ed. The parameters are $pdfFilePath which is the path of the current document in the process, and $pdftotextPath is the path of the library I used for extraction of the text.



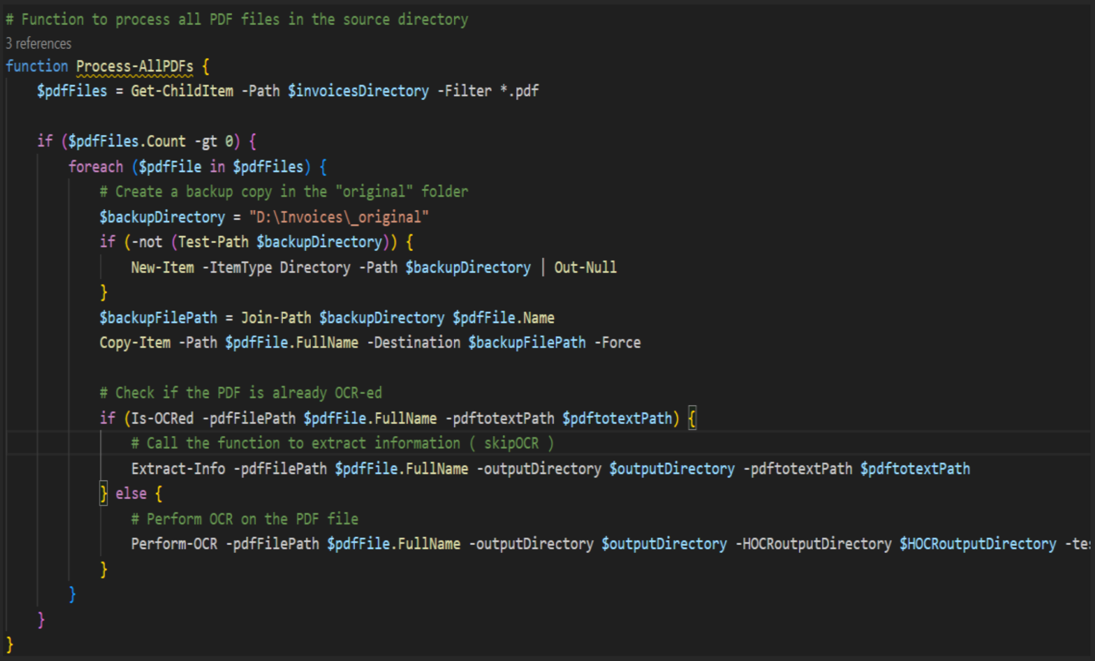
```
# Function to check if PDF is already OCR-ed
2 references
function Is-OCRed {
    param (
        [string]$pdfFilePath,
        [string]$pdftotextPath
    )

    # Extract text from PDF using pdftotext
    $textContent = & $pdftotextPath -q -nopgbrk -eol unix $pdfFilePath -

    # Check if the extracted text is non-empty
    if ($null -ne $textContent -and $textContent.Trim()) {
        return $true
    } else {
        return $false
    }
}
```

SkipOCR Function:

If the document is found to be already OCR-ed, the system bypasses the OCR process and directly extracts text using `pdftotext`. The extracted text is then passed to another function that processes it to extract relevant information such as invoice numbers, issue dates, and supplier details.

Process-AllPDFs Function:

This function iterates over all PDF files in the specified directory. For each file, it first creates a backup and then checks if the file is already OCR-ed. Depending on the result, it either extracts the information directly or performs OCR and after that extracting the information.



```powershell
# Function to process all PDF files in the source directory
3 references
function Process-AllPDFs {
    $pdfFiles = Get-ChildItem -Path $invoicesDirectory -Filter *.pdf

    if ($pdfFiles.Count -gt 0) {
        foreach ($pdfFile in $pdfFiles) {
            # Create a backup copy in the "original" folder
            $backupDirectory = "D:\Invoices\_original"
            if (-not (Test-Path $backupDirectory)) {
                New-Item -ItemType Directory -Path $backupDirectory | Out-Null
            }
            $backupFilePath = Join-Path $backupDirectory $pdfFile.Name
            Copy-Item -Path $pdfFile.FullName -Destination $backupFilePath -Force

            # Check if the PDF is already OCR-ed
            if (Is-OCRed -pdfFilePath $pdfFile.FullName -pdftotextPath $pdftotextPath) {
                # Call the function to extract information ( skipOCR )
                Extract-Info -pdfFilePath $pdfFile.FullName -outputDirectory $outputDirectory -pdftotextPath $pdftotextPath
            } else {
                # Perform OCR on the PDF file
                Perform-OCR -pdfFilePath $pdfFile.FullName -outputDirectory $outputDirectory -HOCRoutputDirectory $HOCRoutputDirectory -te
            }
        }
    }
}
```

PDF Text Extraction Process:

Text extraction for digital documents is handled by `pdftotext`, which converts the entire PDF content into plain text. This approach preserves the format and ensures that no data is lost or corrupted.

CSV Files: Suppliers and Journal:

Two CSV files are integral to the system:

1. **Suppliers CSV**:

   Contains supplier information, including multiple keywords to identify relevant details in the text.

```
SupplierName;FriendlyName;InvoiceCode;DateKey;DateKeyDirection;InvKey;InvKeyDir;InvNumLength;
DateFormat;SupplierCUI;SupplierREG
```

2. **Journal CSV**:

   Logs the extracted information, including the supplier name, invoice number, issue date, and other key data.

```
2024-08-19";"D:\Invoices\_ocr\2024-07-16_ORANGE_COMMUNICATIONS_RO9010105_TKR-
240305324193_OCR.pdf";"ORANGE_COMMUNICATIONS";"RO9010105";"J40/8926/1997";"2024-07-16";"TKR-
";"240305324193"
```

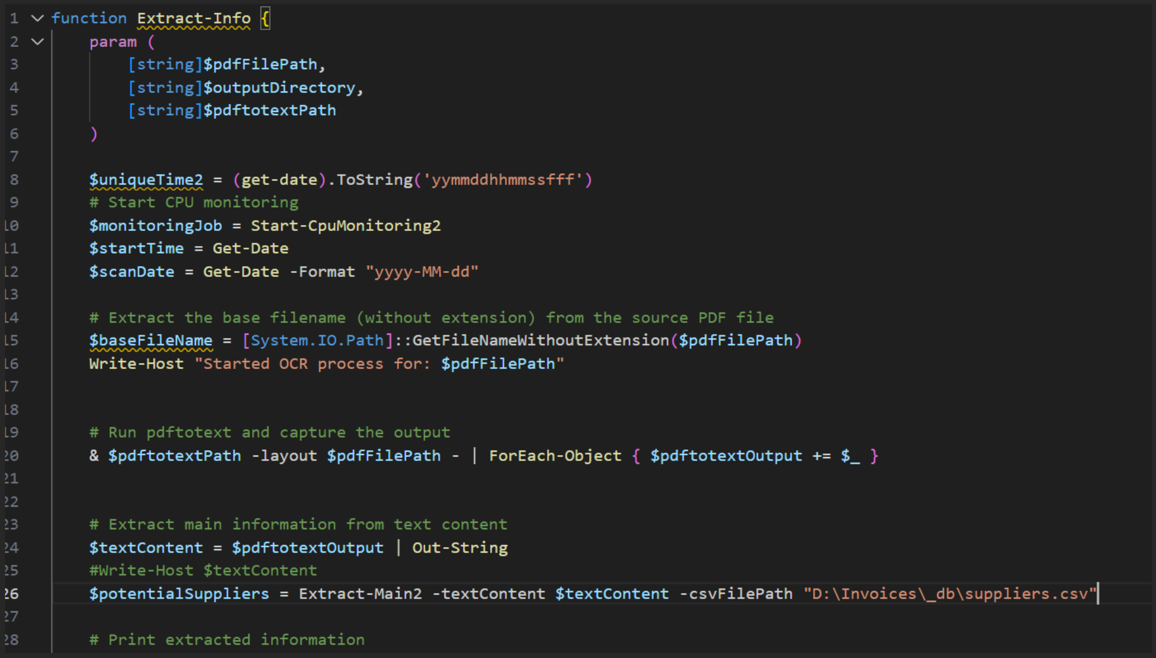
After processing the documents, they are renamed with the name of the information that has been extracted , for easily searching/finding of a specific document.

Keyword Search and Information Extraction:

The system searches for predefined keywords in the extracted text to locate and extract relevant information. This process includes handling various date formats and supplier-specific codes.

skipOCR function Workflow: **Functions Details:**

1) Extract-Info:



```
function Extract-Info {
    param (
        [string]$pdfFilePath,
        [string]$outputDirectory,
        [string]$pdftotextPath
    )

    $uniqueTime2 = (get-date).ToString('yymmddhhmmssfff')
    # Start CPU monitoring
    $monitoringJob = Start-CpuMonitoring2
    $startTime = Get-Date
    $scanDate = Get-Date -Format "yyyy-MM-dd"

    # Extract the base filename (without extension) from the source PDF file
    $baseFileName = [System.IO.Path]::GetFileNameWithoutExtension($pdfFilePath)
    Write-Host "Started OCR process for: $pdfFilePath"


    # Run pdftotext and capture the output
    & $pdftotextPath -layout $pdfFilePath - | ForEach-Object { $pdftotextOutput += $_ }


    # Extract main information from text content
    $textContent = $pdftotextOutput | Out-String
    #Write-Host $textContent
    $potentialSuppliers = Extract-Main2 -textContent $textContent -csvFilePath "D:\Invoices\_db\suppliers.csv"

    # Print extracted information
```

- **Purpose:**
  - Orchestrates the entire extraction process for invoice information from a PDF file.
  - Renames the PDF based on the extracted data and logs the information into a CSV file.

- **Parameters:**

- `$pdfFilePath` (string): The full path of the PDF file to process.
- `$outputDirectory` (string): The directory where processed PDF files will be saved.
- `$pdftotextPath` (string): Path to the `pdftotext` executable for extracting text from the PDF.

- **Workflow:**

1. **CPU Monitoring:**
   - Starts CPU monitoring to measure resource usage during the process.

2. **Text Extraction:**
   - Utilizes `pdftotext` to extract text from the PDF.

3. **Data Extraction:**
   - Extracts relevant information such as supplier name, invoice number, and issue date using the `Extract-Main2` function.

4. **File Renaming:**
   - Renames the PDF file based on the extracted invoice data.

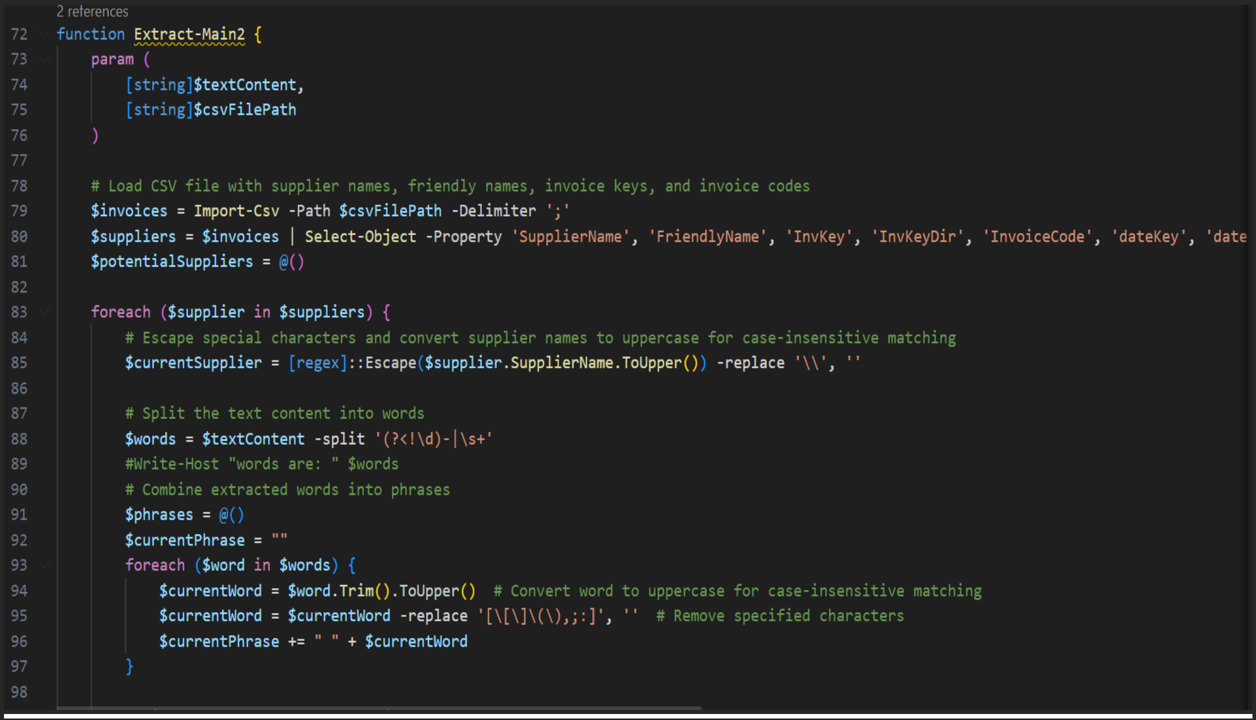
5. **Journal Entry:**
   - Logs the extracted information into a CSV journal.

6. **CPU Monitoring Summary:**
   - Outputs the average CPU usage during the process.

- **Outputs:**

- Renamed PDF file.
- Updated CSV journal with extracted invoice information.

2) ***Extract-Main2***



```
2 references
function Extract-Main2 {
    param (
        [string]$textContent,
        [string]$csvFilePath
    )

    # Load CSV file with supplier names, friendly names, invoice keys, and invoice codes
    $invoices = Import-Csv -Path $csvFilePath -Delimiter ';'
    $suppliers = $invoices | Select-Object -Property 'SupplierName', 'FriendlyName', 'InvKey', 'InvKeyDir', 'InvoiceCode', 'dateKey', 'date
    $potentialSuppliers = @()

    foreach ($supplier in $suppliers) {
        # Escape special characters and convert supplier names to uppercase for case-insensitive matching
        $currentSupplier = [regex]::Escape($supplier.SupplierName.ToUpper()) -replace '\\', ''

        # Split the text content into words
        $words = $textContent -split '(?<!\d)-|\s+'
        #Write-Host "words are: " $words
        # Combine extracted words into phrases
        $phrases = @()
        $currentPhrase = ""
        foreach ($word in $words) {
            $currentWord = $word.Trim().ToUpper()  # Convert word to uppercase for case-insensitive matching
            $currentWord = $currentWord -replace '[\[\]\(\),;:]', ''  # Remove specified characters
            $currentPhrase += " " + $currentWord
        }

```

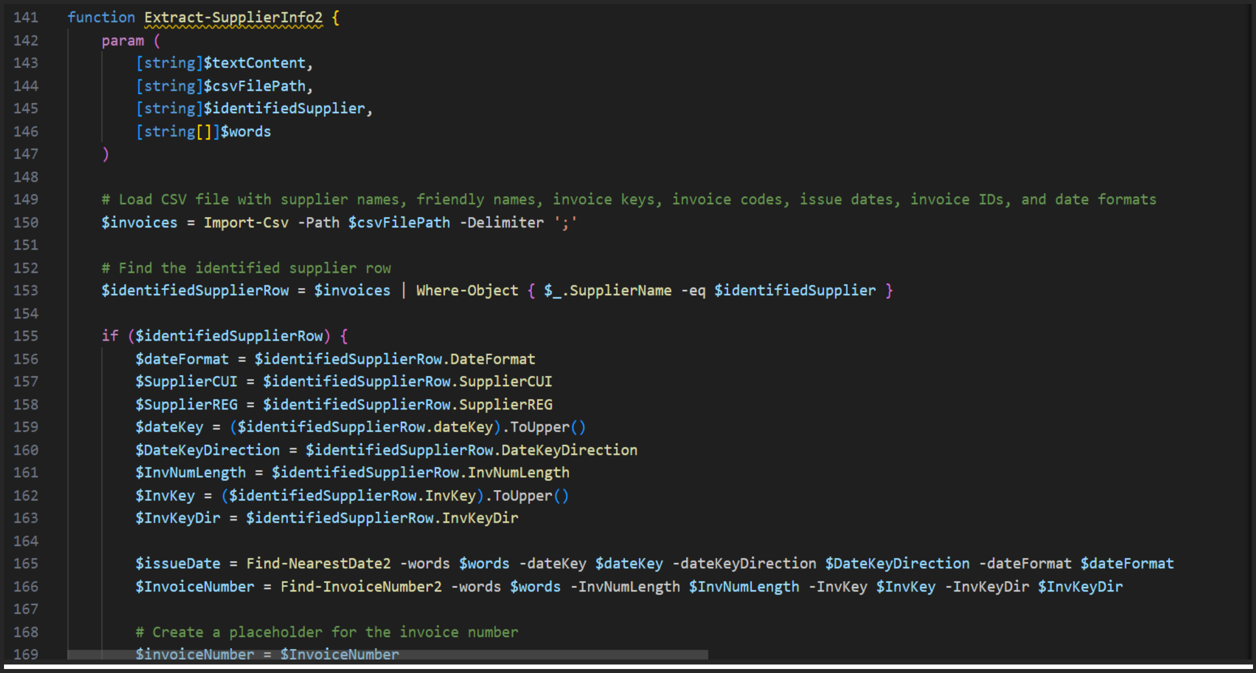
- **Purpose:**
  - Identifies the supplier in the text content and extracts invoice-related data.

- **Parameters:**
  - `$textContent` (string): The extracted text content from the PDF.

  - `$csvFilePath` (string): Path to the CSV file containing supplier information and keys.

- **Workflow:**
    1. **CSV Loading:**
        - Loads supplier information from the specified CSV file.
    2. **Supplier Matching:**
        - Matches suppliers based on a pre-defined pattern in the text content.
    3. **Data Extraction:**
        - Calls `Extract-SupplierInfo2` to extract specific details like invoice number and issue date.
    4. **Fallback:**
        - Returns placeholder data if no match is found.
- **Outputs:**
    - An object containing the matched supplier's information, or placeholder data if no match is found.

3) ***Extract-SupplierInfo2***



```powershell
function Extract-SupplierInfo2 {
    param (
        [string]$textContent,
        [string]$csvFilePath,
        [string]$identifiedSupplier,
        [string[]]$words
    )

    # Load CSV file with supplier names, friendly names, invoice keys, invoice codes, issue dates, invoice IDs, and date formats
    $invoices = Import-Csv -Path $csvFilePath -Delimiter ';'

    # Find the identified supplier row
    $identifiedSupplierRow = $invoices | Where-Object { $_.SupplierName -eq $identifiedSupplier }

    if ($identifiedSupplierRow) {
        $dateFormat = $identifiedSupplierRow.DateFormat
        $SupplierCUI = $identifiedSupplierRow.SupplierCUI
        $SupplierREG = $identifiedSupplierRow.SupplierREG
        $dateKey = ($identifiedSupplierRow.dateKey).ToUpper()
        $DateKeyDirection = $identifiedSupplierRow.DateKeyDirection
        $InvNumLength = $identifiedSupplierRow.InvNumLength
        $InvKey = ($identifiedSupplierRow.InvKey).ToUpper()
        $InvKeyDir = $identifiedSupplierRow.InvKeyDir

        $issueDate = Find-NearestDate2 -words $words -dateKey $dateKey -dateKeyDirection $DateKeyDirection -dateFormat $dateFormat
        $InvoiceNumber = Find-InvoiceNumber2 -words $words -InvNumLength $InvNumLength -InvKey $InvKey -InvKeyDir $InvKeyDir

        # Create a placeholder for the invoice number
        $invoiceNumber = $InvoiceNumber
```

- **Purpose:**
    - Extracts specific supplier-related information such as invoice numbers and issue dates.

- **Parameters:**
    - `$textContent` (string): The full text content from the PDF.
    - `$csvFilePath` (string): Path to the CSV file with supplier details.
    - `$identifiedSupplier` (string): The name of the supplier identified in the text.
    - `$words` (string[]): The words extracted and processed from the text content.

- **Workflow:**

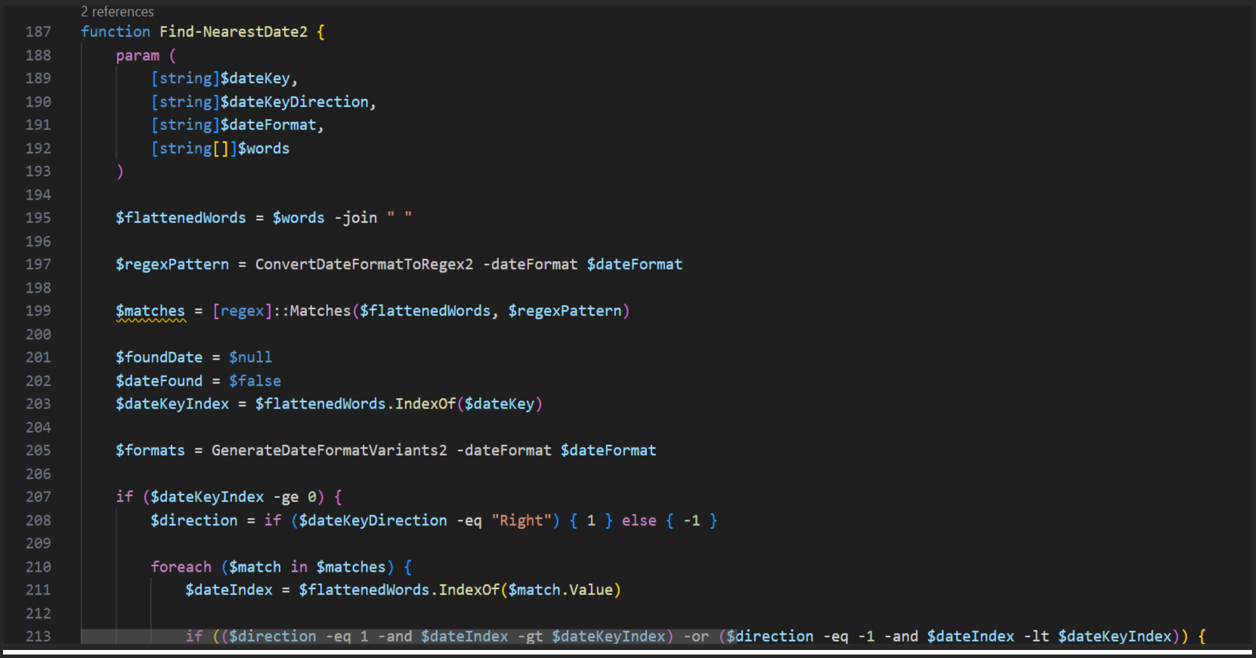
  1. **CSV Row Matching:**
     - Identifies the correct supplier row in the CSV based on the supplier name.

  2. **Invoice and Date Extraction:**
     - Uses `Find-InvoiceNumber2` and `Find-NearestDate2` to extract the invoice number and issue date.

  3. **Return Data:**
     - Creates and returns an object containing the extracted supplier information.

- **Outputs:**
  - A PowerShell object containing extracted supplier details.

4) ***Find-NearestDate2***



```
2 references
function Find-NearestDate2 {
    param (
        [string]$dateKey,
        [string]$dateKeyDirection,
        [string]$dateFormat,
        [string[]]$words
    )

    $flattenedWords = $words -join " "

    $regexPattern = ConvertDateFormatToRegex2 -dateFormat $dateFormat

    $matches = [regex]::Matches($flattenedWords, $regexPattern)

    $foundDate = $null
    $dateFound = $false
    $dateKeyIndex = $flattenedWords.IndexOf($dateKey)

    $formats = GenerateDateFormatVariants2 -dateFormat $dateFormat

    if ($dateKeyIndex -ge 0) {
        $direction = if ($dateKeyDirection -eq "Right") { 1 } else { -1 }

        foreach ($match in $matches) {
            $dateIndex = $flattenedWords.IndexOf($match.Value)

            if (($direction -eq 1 -and $dateIndex -gt $dateKeyIndex) -or ($direction -eq -1 -and $dateIndex -lt $dateKeyIndex)) {
```

- **Purpose:**
    - Finds and extracts the date nearest to a specified keyword in the text.

- **Parameters:**
    - `$dateKey` (string): The keyword to locate near which the date will be extracted.
    - `$dateKeyDirection` (string): Direction (`Right` or `Left`) to search relative to the `dateKey`.
    - `$dateFormat` (string): Expected date format (e.g., `yyyy-MM-dd`).
    - `$words` (string[]): Array of words extracted from the text.

- **Workflow:**

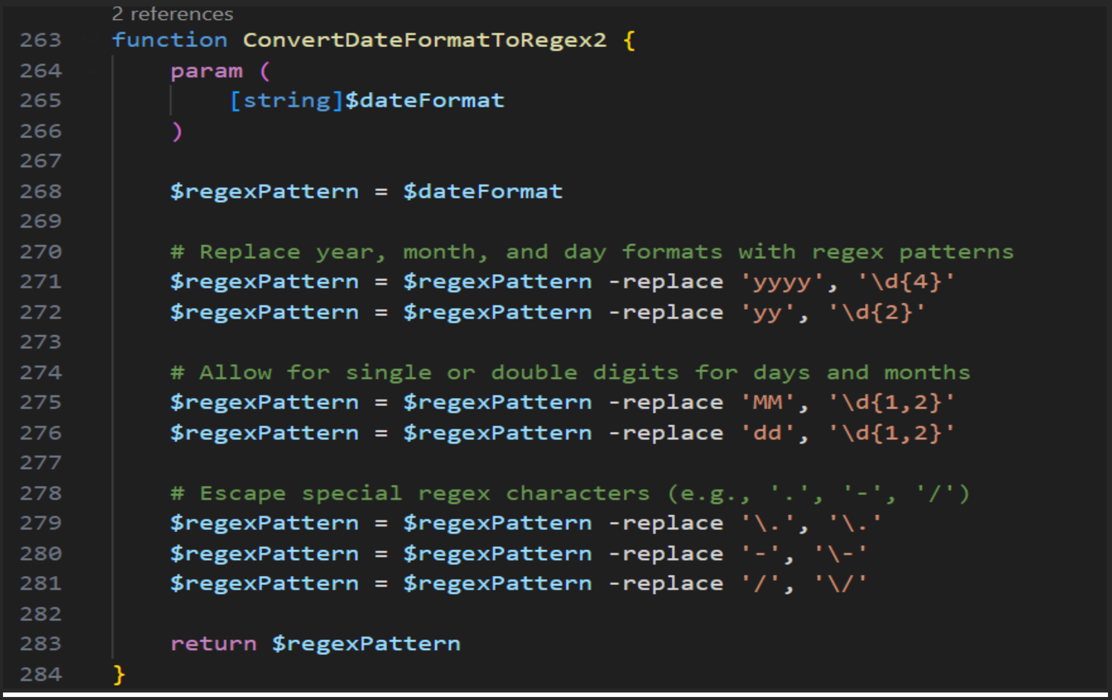
  1. **Flatten Words:**
     - Joins the words into a single string for processing.

  2. **Pattern Matching:**
     - Converts the expected date format into a regex pattern using `ConvertDateFormatToRegex2`.

  3. **Date Extraction:**
     - Identifies and validates the date based on the keyword's position and the regex pattern.

- **Outputs:**
  - The nearest date found in the specified format or `null` if no valid date is found.

5) ***ConvertDateFormatToRegex2***



```
2 references
function ConvertDateFormatToRegex2 {
    param (
        [string]$dateFormat
    )

    $regexPattern = $dateFormat

    # Replace year, month, and day formats with regex patterns
    $regexPattern = $regexPattern -replace 'yyyy', '\d{4}'
    $regexPattern = $regexPattern -replace 'yy', '\d{2}'

    # Allow for single or double digits for days and months
    $regexPattern = $regexPattern -replace 'MM', '\d{1,2}'
    $regexPattern = $regexPattern -replace 'dd', '\d{1,2}'

    # Escape special regex characters (e.g., '.', '-', '/')
    $regexPattern = $regexPattern -replace '\.', '\.'
    $regexPattern = $regexPattern -replace '-', '\-'
    $regexPattern = $regexPattern -replace '/', '\/'

    return $regexPattern
}
```

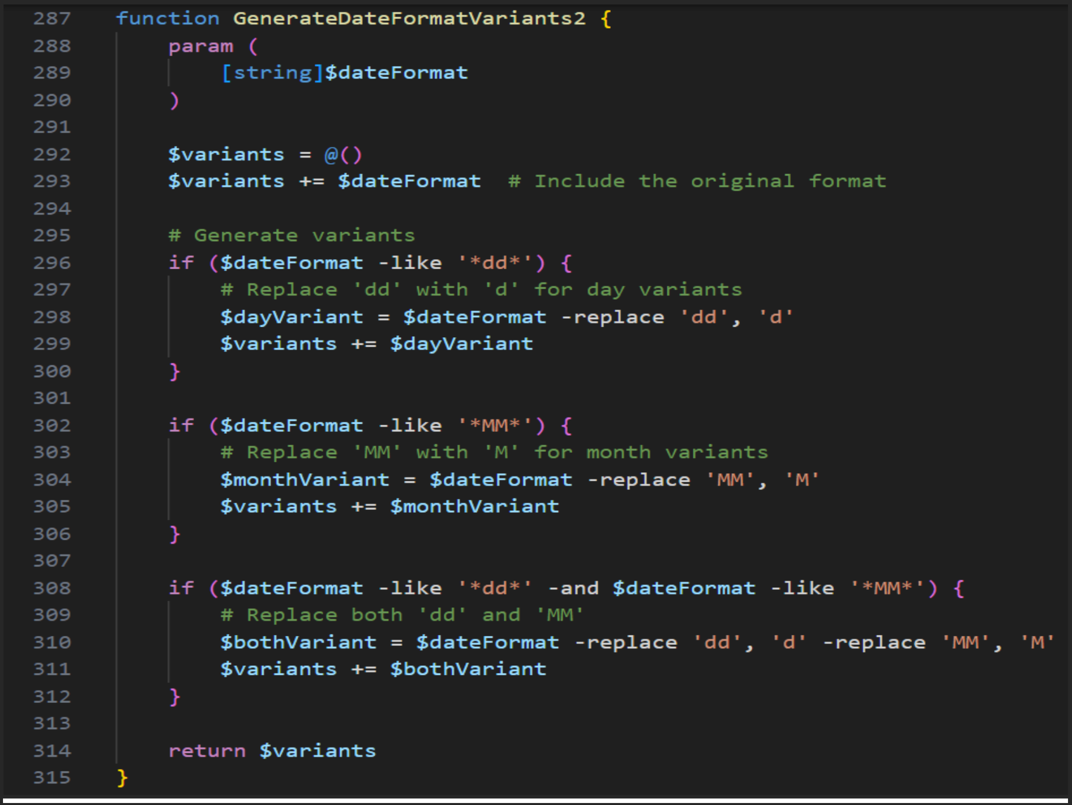
- **Purpose:**
    - Converts a given date format into a regex pattern for date extraction.

- **Parameters:**
    - `$dateFormat` (string): The expected date format (e.g., `yyyy-MM-dd`).

- **Workflow:**
    1. **Replace Placeholders:**
        - Replaces date format placeholders with equivalent regex patterns.
    2. **Handle Special Characters:**
        - Escapes special characters used in the date format.

- **Outputs:**
    - A regex pattern string that matches the expected date format.

6) ***GenerateDateFormatVariants2***



```powershell
function GenerateDateFormatVariants2 {
    param (
        [string]$dateFormat
    )

    $variants = @()
    $variants += $dateFormat  # Include the original format

    # Generate variants
    if ($dateFormat -like '*dd*') {
        # Replace 'dd' with 'd' for day variants
        $dayVariant = $dateFormat -replace 'dd', 'd'
        $variants += $dayVariant
    }

    if ($dateFormat -like '*MM*') {
        # Replace 'MM' with 'M' for month variants
        $monthVariant = $dateFormat -replace 'MM', 'M'
        $variants += $monthVariant
    }

    if ($dateFormat -like '*dd*' -and $dateFormat -like '*MM*') {
        # Replace both 'dd' and 'MM'
        $bothVariant = $dateFormat -replace 'dd', 'd' -replace 'MM', 'M'
        $variants += $bothVariant
    }

    return $variants
}
```

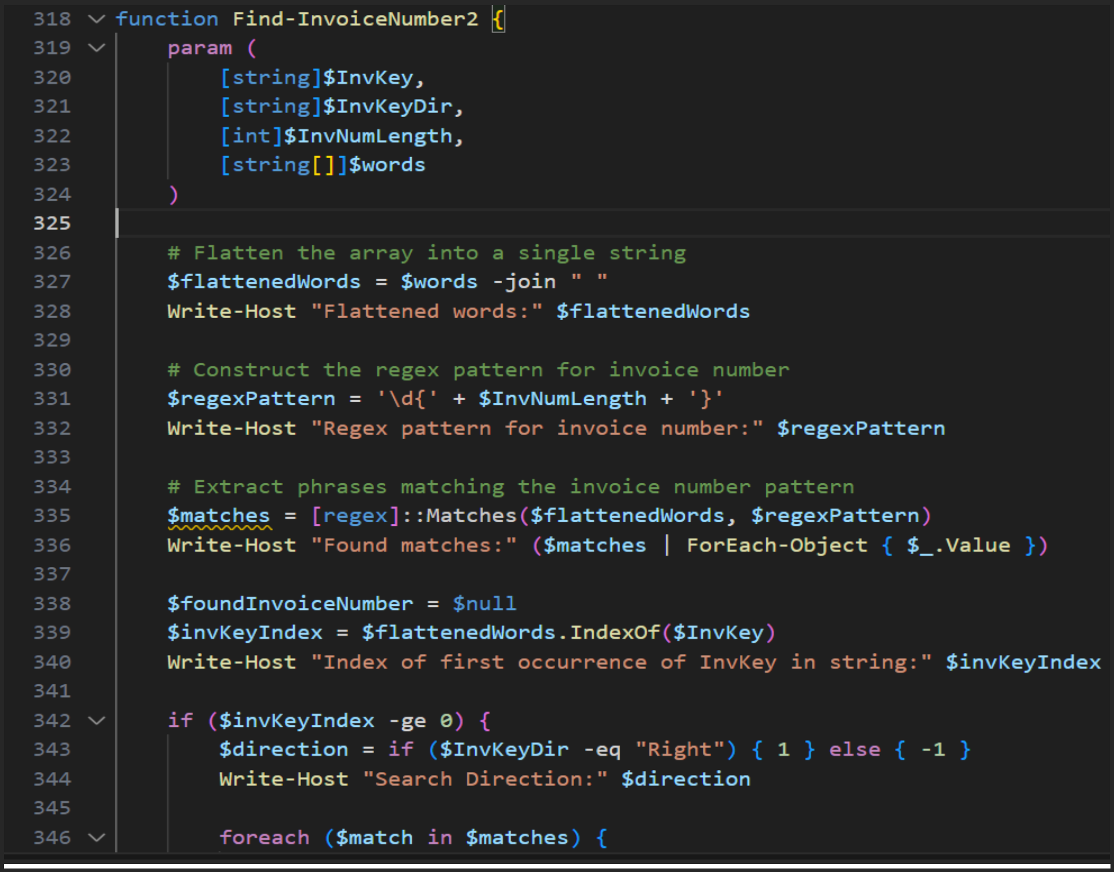
- **Purpose:**
    - Generates variations of a given date format to account for differences in how dates might be presented.

- **Parameters:**
    - `$dateFormat` (string): The original date format.

- **Workflow:**
    1. **Generate Variants:**
        - Creates variations by substituting parts of the format (e.g., `dd` to `d`, `MM` to `M`).

- **Outputs:**
    - An array of possible date format variants.

7) Find-InvoiceNumber2



```
function Find-InvoiceNumber2 {
    param (
        [string]$InvKey,
        [string]$InvKeyDir,
        [int]$InvNumLength,
        [string[]]$words
    )

    # Flatten the array into a single string
    $flattenedWords = $words -join " "
    Write-Host "Flattened words:" $flattenedWords

    # Construct the regex pattern for invoice number
    $regexPattern = '\d{' + $InvNumLength + '}'
    Write-Host "Regex pattern for invoice number:" $regexPattern

    # Extract phrases matching the invoice number pattern
    $matches = [regex]::Matches($flattenedWords, $regexPattern)
    Write-Host "Found matches:" ($matches | ForEach-Object { $_.Value })

    $foundInvoiceNumber = $null
    $invKeyIndex = $flattenedWords.IndexOf($InvKey)
    Write-Host "Index of first occurrence of InvKey in string:" $invKeyIndex

    if ($invKeyIndex -ge 0) {
        $direction = if ($InvKeyDir -eq "Right") { 1 } else { -1 }
        Write-Host "Search Direction:" $direction

        foreach ($match in $matches) {
```

- **Purpose:**
  - Extracts the invoice number from the text using a key pattern.

- **Parameters:**
  - `$InvKey` (string): The key string near which the invoice number is expected.
  - `$InvKeyDir` (string): Direction (`Right` or `Left`) to search for the invoice number relative to the `InvKey`.
  - `$InvNumLength` (int): Expected length of the invoice number.
  - `$words` (string[]): Array of words extracted from the text.

- **Workflow:**

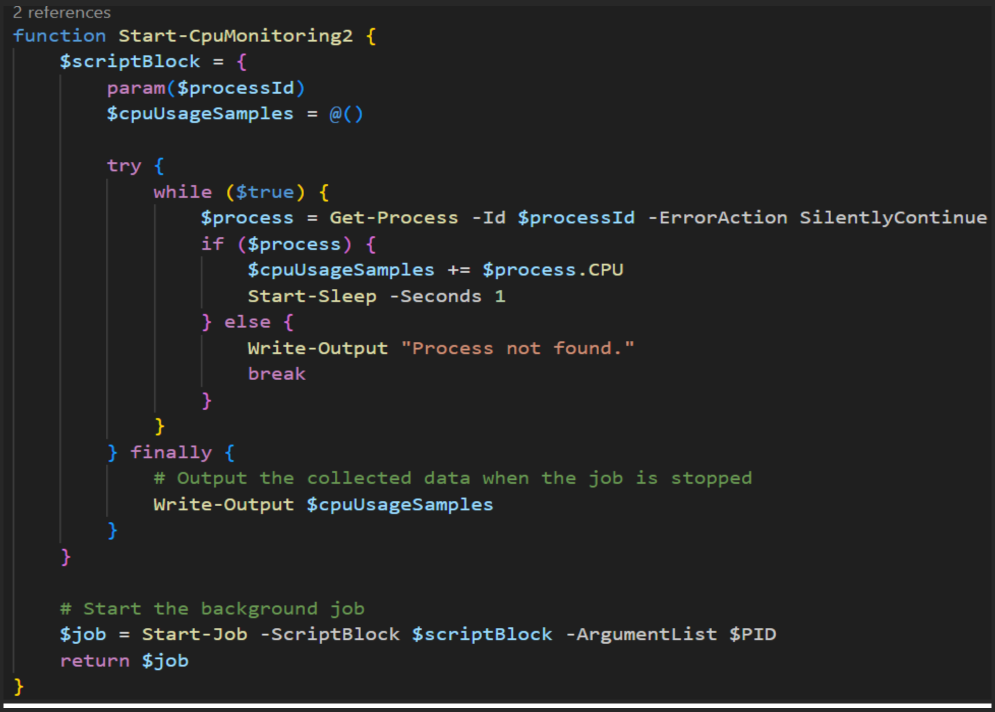
1. **Flatten Words:**
    - Joins the words into a single string for easier processing.

2. **Pattern Matching:**
    - Uses a regex pattern based on the expected length of the invoice number to find potential matches.

3. **Extract Invoice Number:**
    - Identifies the invoice number based on its proximity to the key string.

- **Outputs:**
    - The found invoice number or `null` if no match is found.

8) ***Start-CpuMonitoring2 & Stop-CpuMonitoring***



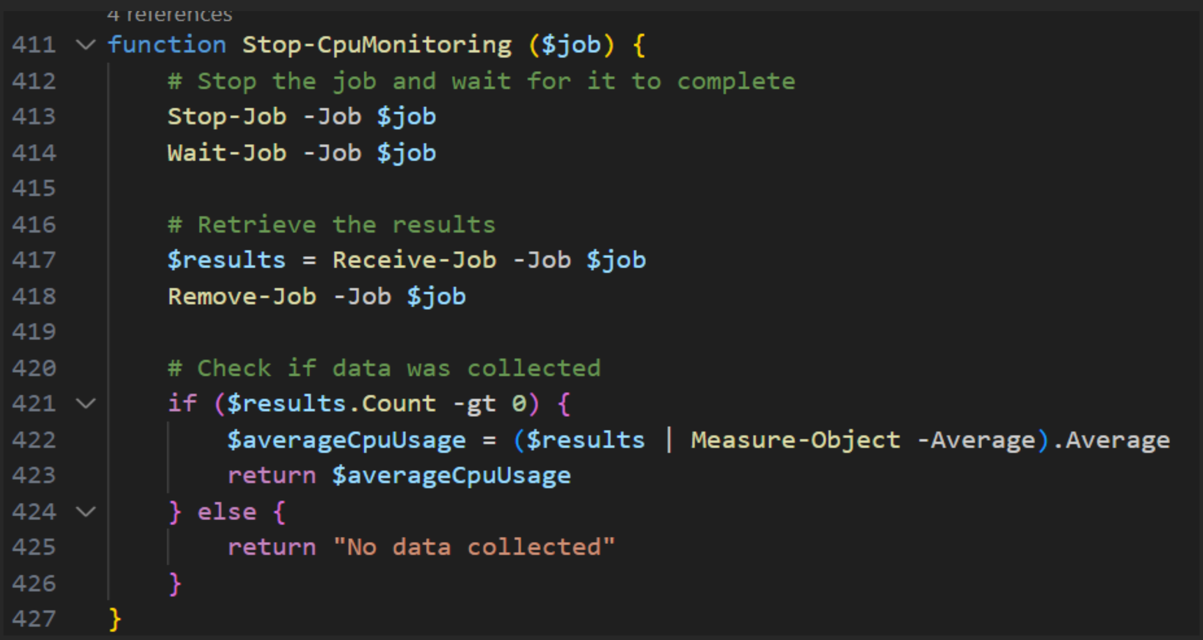
```
2 references
function Start-CpuMonitoring2 {
    $scriptBlock = {
        param($processId)
        $cpuUsageSamples = @()

        try {
            while ($true) {
                $process = Get-Process -Id $processId -ErrorAction SilentlyContinue
                if ($process) {
                    $cpuUsageSamples += $process.CPU
                    Start-Sleep -Seconds 1
                } else {
                    Write-Output "Process not found."
                    break
                }
            }
        } finally {
            # Output the collected data when the job is stopped
            Write-Output $cpuUsageSamples
        }
    }

    # Start the background job
    $job = Start-Job -ScriptBlock $scriptBlock -ArgumentList $PID
    return $job
}
```

- **Purpose:**
    - Monitors the CPU usage during the extraction process.

- **Parameters for `Start-CpuMonitoring2`:**
    - None directly; it uses the current process ID.

- **Workflow:**
    1. **CPU Monitoring:**
        - Continuously records CPU usage until stopped.

- **Outputs:**
    - A job object used for monitoring, which is later stopped and processed by `Stop-CpuMonitoring`.



```
4 references
function Stop-CpuMonitoring ($job) {
    # Stop the job and wait for it to complete
    Stop-Job -Job $job
    Wait-Job -Job $job

    # Retrieve the results
    $results = Receive-Job -Job $job
    Remove-Job -Job $job

    # Check if data was collected
    if ($results.Count -gt 0) {
        $averageCpuUsage = ($results | Measure-Object -Average).Average
        return $averageCpuUsage
    } else {
        return "No data collected"
    }
}
```

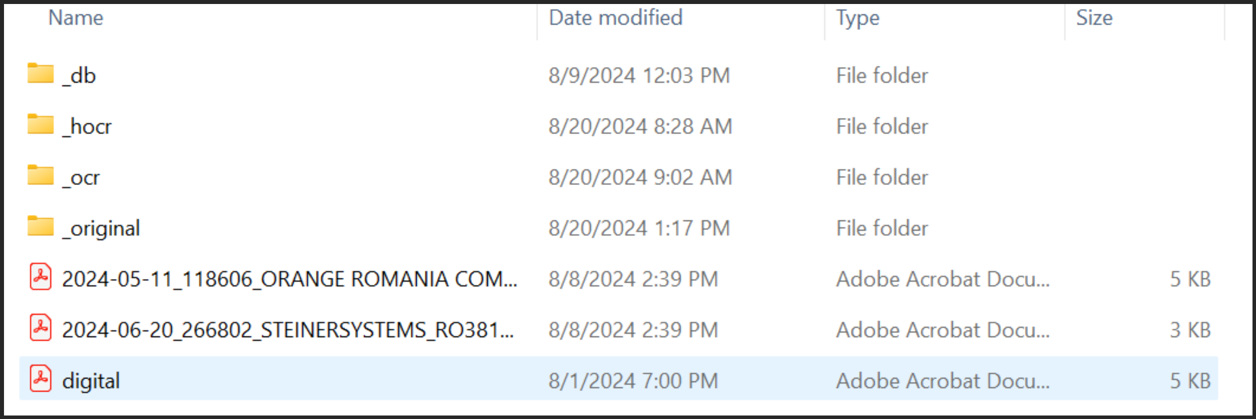
- **Parameters for `Stop-CpuMonitoring`:**
  - `$monitoringJob` (object): The job object returned by `Start-CpuMonitoring2`.

- **Workflow:**
  1. **Job Completion:**
     - Stops the CPU monitoring job.
  2. **CPU Data Calculation:**
     - Computes the average CPU usage during the monitored period.

- **Outputs:**
  - Displays the average CPU usage in percentage.

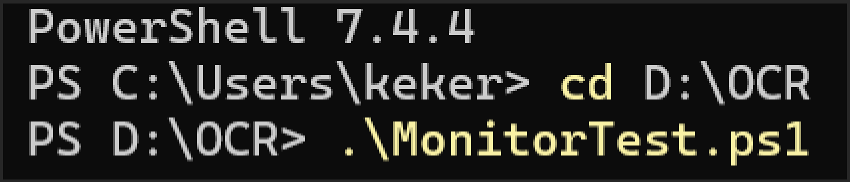
How the script works:

1) Put all the pdf documents that you want in the “Invoices” folder. The script will automatically process every pdf from this folder.



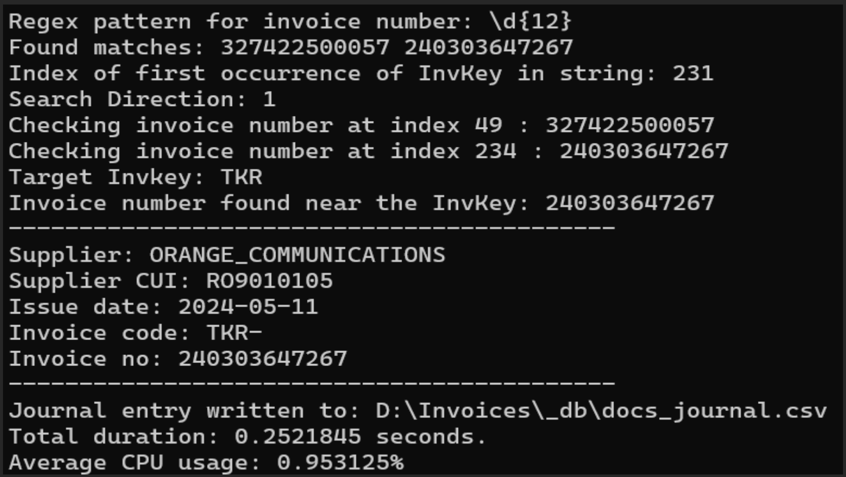
Name	Date modified	Type	Size
_db	8/9/2024 12:03 PM	File folder	
_hocr	8/20/2024 8:28 AM	File folder	
_ocr	8/20/2024 9:02 AM	File folder	
_original	8/20/2024 1:17 PM	File folder	
2024-05-11_118606_ORANGE ROMANIA COM...	8/8/2024 2:39 PM	Adobe Acrobat Docu...	5 KB
2024-06-20_266802_STEINERSYSTEMS_RO381...	8/8/2024 2:39 PM	Adobe Acrobat Docu...	3 KB
digital	8/1/2024 7:00 PM	Adobe Acrobat Docu...	5 KB

2) Open powershell or cmd and run the script:

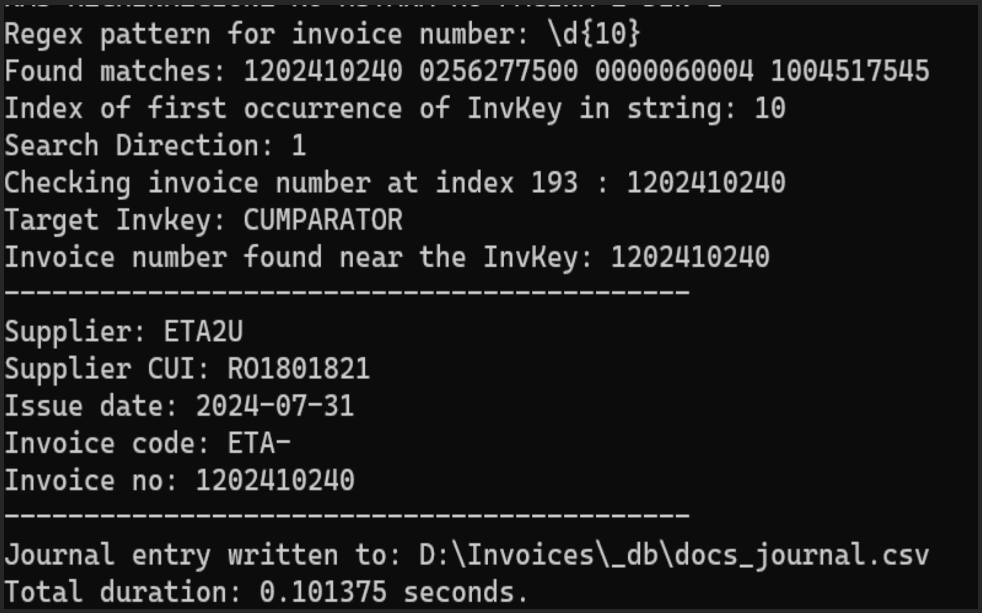


```
PowerShell 7.4.4
PS C:\Users\keker> cd D:\OCR
PS D:\OCR> .\MonitorTest.ps1
```

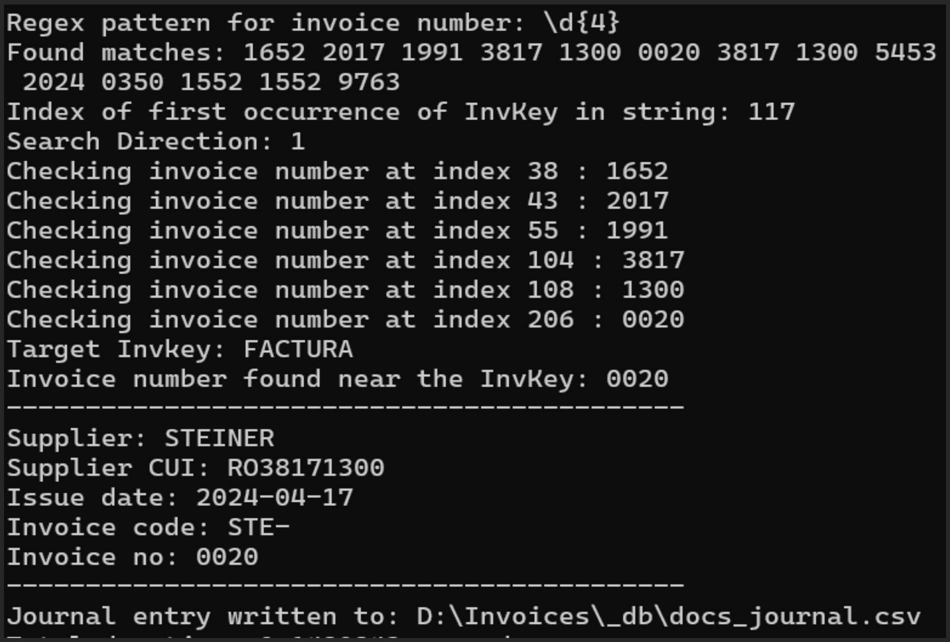
3) The output is the extracted text and the extracted information about the invoice:



```
Regex pattern for invoice number: \d{12}
Found matches: 327422500057 240303647267
Index of first occurrence of InvKey in string: 231
Search Direction: 1
Checking invoice number at index 49 : 327422500057
Checking invoice number at index 234 : 240303647267
Target Invkey: TKR
Invoice number found near the InvKey: 240303647267
------------------------------------------
Supplier: ORANGE_COMMUNICATIONS
Supplier CUI: RO9010105
Issue date: 2024-05-11
Invoice code: TKR-
Invoice no: 240303647267
------------------------------------------
Journal entry written to: D:\Invoices\_db\docs_journal.csv
Total duration: 0.2521845 seconds.
Average CPU usage: 0.953125%
```



```
[illegible]
Regex pattern for invoice number: \d{10}
Found matches: 1202410240 0256277500 0000060004 1004517545
Index of first occurrence of InvKey in string: 10
Search Direction: 1
Checking invoice number at index 193 : 1202410240
Target Invkey: CUMPARATOR
Invoice number found near the InvKey: 1202410240
-------------------------------------------
Supplier: ETA2U
Supplier CUI: RO1801821
Issue date: 2024-07-31
Invoice code: ETA-
Invoice no: 1202410240
-------------------------------------------
Journal entry written to: D:\Invoices\_db\docs_journal.csv
Total duration: 0.101375 seconds.
```



```
Regex pattern for invoice number: \d{4}
Found matches: 1652 2017 1991 3817 1300 0020 3817 1300 5453
 2024 0350 1552 1552 9763
Index of first occurrence of InvKey in string: 117
Search Direction: 1
Checking invoice number at index 38 : 1652
Checking invoice number at index 43 : 2017
Checking invoice number at index 55 : 1991
Checking invoice number at index 104 : 3817
Checking invoice number at index 108 : 1300
Checking invoice number at index 206 : 0020
Target Invkey: FACTURA
Invoice number found near the InvKey: 0020
-------------------------------------------
Supplier: STEINER
Supplier CUI: RO38171300
Issue date: 2024-04-17
Invoice code: STE-
Invoice no: 0020
-------------------------------------------
Journal entry written to: D:\Invoices\_db\docs_journal.csv
[illegible]
```

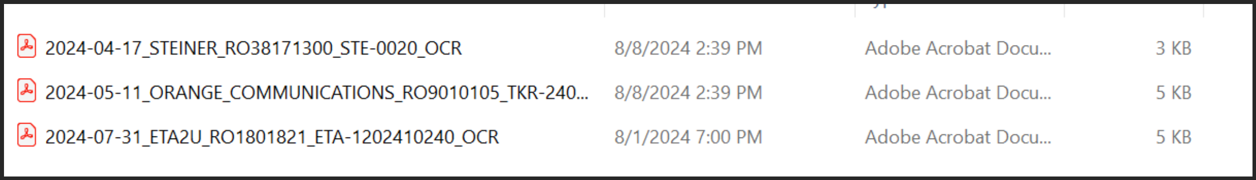
The following data was inserted in the invoices journal :

```
"2024-08-20";"D:\Invoices\_ocr\2024-05-11_ORANGE_COMMUNICATIONS_RO9010105_TKR-
240303647267_OCR.pdf";"ORANGE_COMMUNICATIONS";"RO9010105";"J40/8926/1997";"2024-05-11";"TKR-
";"240303647267"
```

```
"2024-08-20";"D:\Invoices\_ocr\2024-04-17_STEINER_RO38171300_STE-
0020_OCR.pdf";"STEINER";"RO38171300";"J26/1652/2017";"2024-04-17";"STE-";"0020"
```

```
"2024-08-20";"D:\Invoices\_ocr\2024-07-31_ETA2U_RO1801821_ETA-
1202410240_OCR.pdf";"ETA2U";"RO1801821";"J35/703/1992";"2024-07-31";"ETA-";"1202410240"
```

Then the pdfs were moved to the processed pdfs folder:

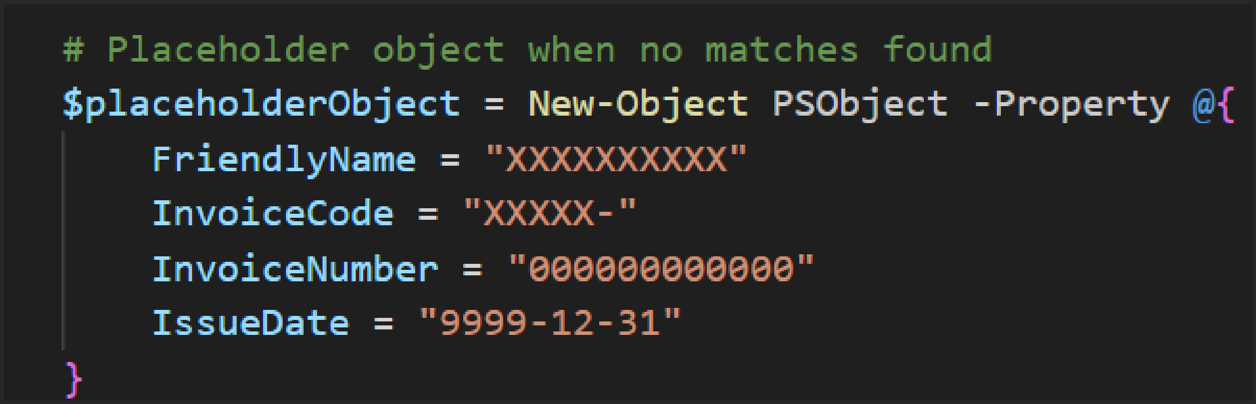


2024-04-17_STEINER_RO38171300_STE-0020_OCR	8/8/2024 2:39 PM	Adobe Acrobat Docu...	3 KB
2024-05-11_ORANGE_COMMUNICATIONS_RO9010105_TKR-240...	8/8/2024 2:39 PM	Adobe Acrobat Docu...	5 KB
2024-07-31_ETA2U_RO1801821_ETA-1202410240_OCR	8/1/2024 7:00 PM	Adobe Acrobat Docu...	5 KB

After processing the documents, they are renamed with the name of the information that has been extracted , for easily searching/finding of a specific document.

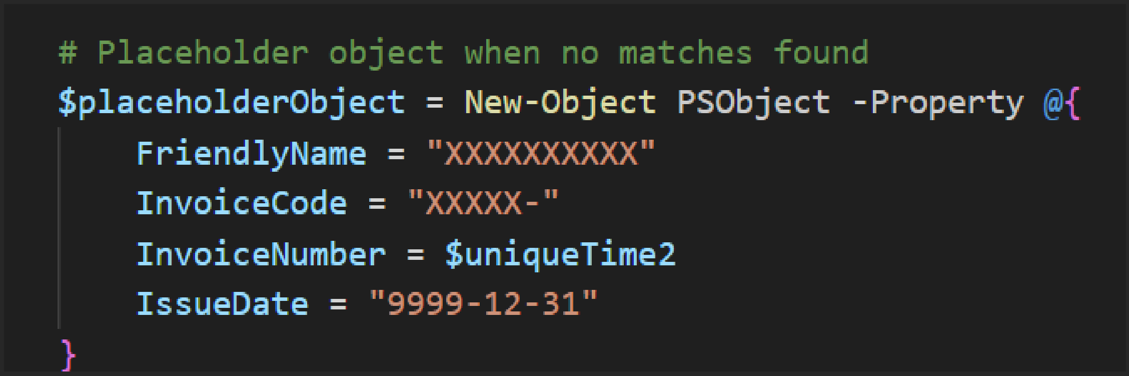
If the information cannot be found and extracted, a placeholder object is defined so the document is renamed with those variables and know that there is a problem with the extraction.

The problem with this is that if we have multiple pdfs that have these values , all of them will have the same name, so after processing them and putting them in the “processed” folder, there will be just the last processed pdf because the rest were overwritten .



```
# Placeholder object when no matches found
$placeholderObject = New-Object PSObject -Property @{
    FriendlyName = "XXXXXXXXXX"
    InvoiceCode = "XXXXX-"
    InvoiceNumber = "000000000000"
    IssueDate = "9999-12-31"
}
```

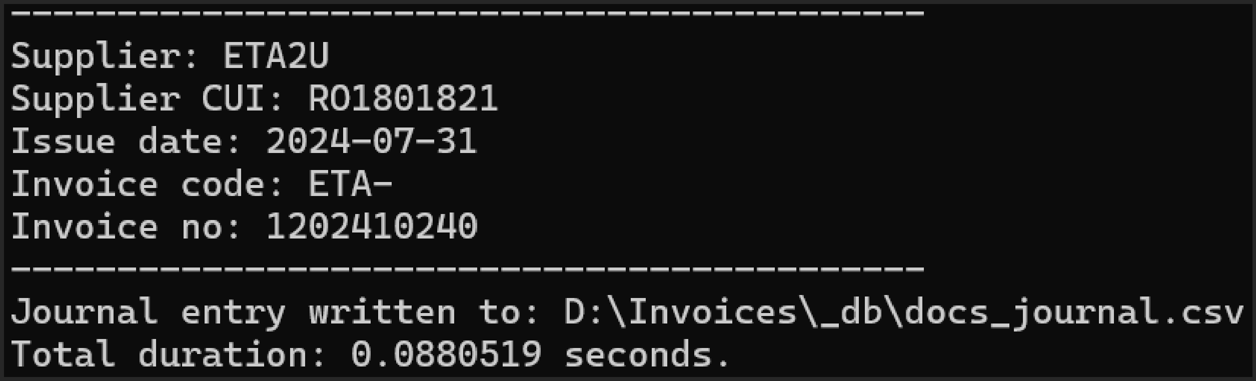
So I modified the placeholder to be unique, handling the situation when we have multiple Pdfs with problems. I added an unique time stamp at the start of the process function , in milliseconds, so at every pdf that is processed there will be a different value. In this way we can store all of the “problem” pdfs and identify them.



```
# Placeholder object when no matches found
$placeholderObject = New-Object PSObject -Property @{
    FriendlyName = "XXXXXXXXXX"
    InvoiceCode = "XXXXX-"
    InvoiceNumber = $uniqueTime2
    IssueDate = "9999-12-31"
}
```

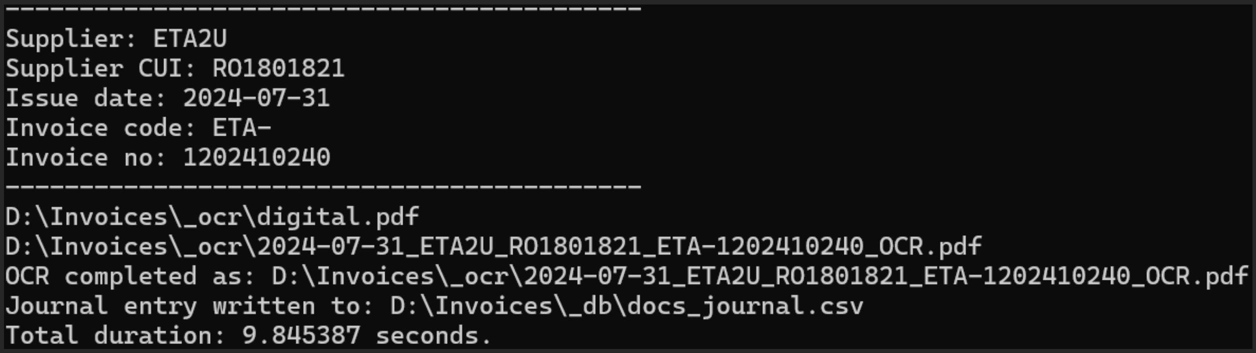
Let’s test the Efficiency of the new function and compare the Total duration for extracting the required informations from the pdfs. I will put the same invoice, one I will run with the new function skipOCR, and the other one I will run with the old script .

The digital pdf (skipOCR) :



```
-------------------------------------------
Supplier: ETA2U
Supplier CUI: RO1801821
Issue date: 2024-07-31
Invoice code: ETA-
Invoice no: 1202410240
-------------------------------------------
Journal entry written to: D:\Invoices\_db\docs_journal.csv
Total duration: 0.0880519 seconds.
```

The digital pdf (old script) :



```
-------------------------------------------
Supplier: ETA2U
Supplier CUI: RO1801821
Issue date: 2024-07-31
Invoice code: ETA-
Invoice no: 1202410240
-------------------------------------------
D:\Invoices\_ocr\digital.pdf
D:\Invoices\_ocr\2024-07-31_ETA2U_RO1801821_ETA-1202410240_OCR.pdf
OCR completed as: D:\Invoices\_ocr\2024-07-31_ETA2U_RO1801821_ETA-1202410240_OCR.pdf
Journal entry written to: D:\Invoices\_db\docs_journal.csv
Total duration: 9.845387 seconds.
```

We can see a difference of 9 seconds. It’s not a big difference if processing just a few documents. But when we have more documents to process then the saved time will be efficient.

OCR Function inherently is prone to mistakes . (scan quality , similar characters miss-recognition example: “O instead of 0” etc.)

But more important is that the new function makes the processing of the digital documents 100% accurately, the text is extracted without mistakes, so there will be no errors from the digital documents.

So I achieved my goal for this optimization project.