AUTOMATED DATA EXTRACTION AND CONVERSION FROM EXCEL FILE TO JSON -HORIZON APPLICATION

AUTOMATION TESTING

***Sindhujaa Guntupalli***

Sept 14, 2022

Version 0.1

Contents

[Converting reports from excel file to JSON file for comparing data in the share module between UI and other export methods. 4](#_Toc114084283)

[Problem Statement: 4](#_Toc114084284)

[Solution: 4](#_Toc114084285)

[Coding details and explanation with steps: 4](#_Toc114084286)

[Execution status 6](#_Toc114084287)

[JSON file is created 6](#_Toc114084288)

[Downloaded Excel File 6](#_Toc114084289)

[Newly generated JSON file from above .xlsx file 7](#_Toc114084290)

[Final Output (JSON file created and comparison result) after executing the Script 7](#_Toc114084291)

**Summary**:

The document provides solution with automated steps to extract reporting data from excel files and convert them to Json format, this data is fetched from the Horizon Application (Share module). Currently, data is compared manually by the testing resource and involves huge manual effort and time taking process.

With the automated function built in JavaScript, the data is extracted and converted to Json files and quickly available for comparison. This solution also provides the comparison results with highlighting any differences/errors occurred in process of data extraction and conversion.

Once implemented this solution will increase the resource efficiency and lead as best practise across the project.

# Converting reports from excel file to JSON file for comparing data in the share module between UI and other export methods.

## Problem Statement:

Converting reports from excel to JSON file for comparing data in the share module between UI and other export methods.

## Solution:

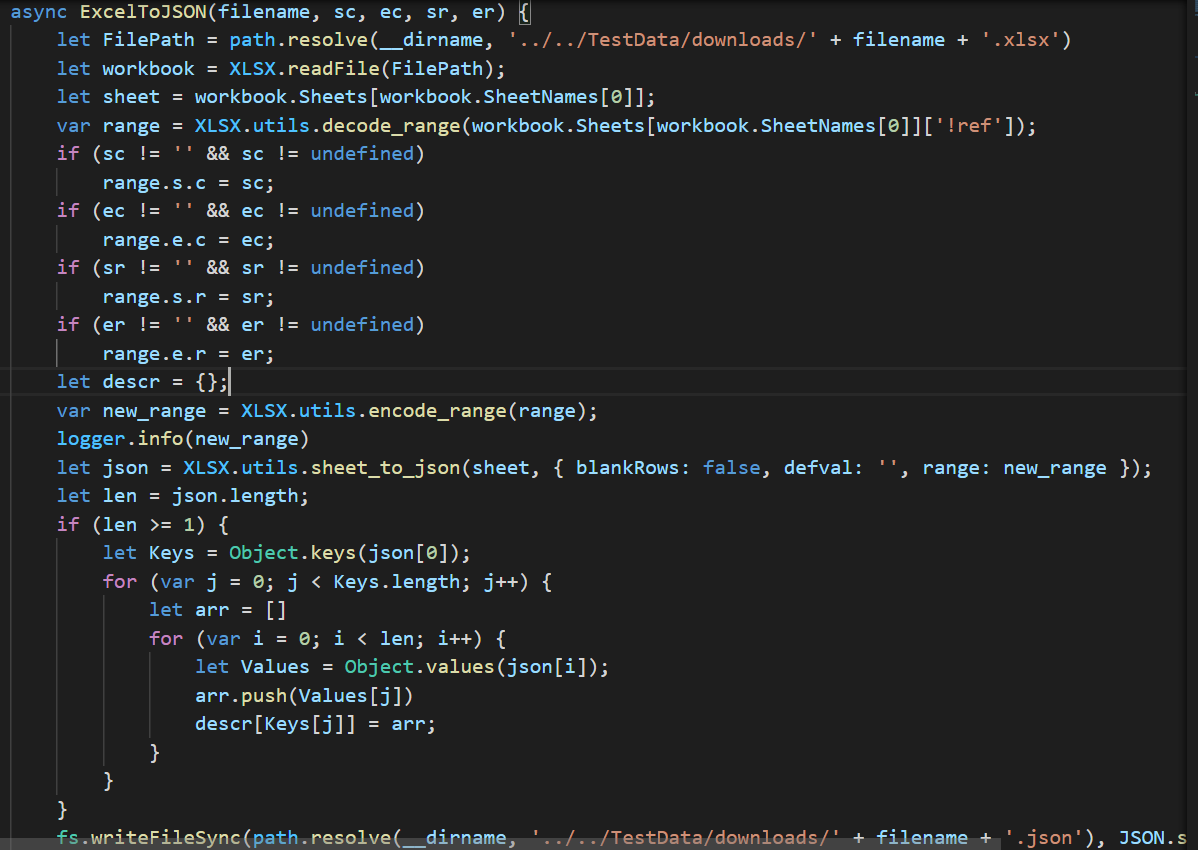
Using this **EXCELTOJSON** function logic.

We can validate all the reports at a time.

|  |  |  |
| --- | --- | --- |
|  |  | **Time Consumption (Minutes)** |
| **AS IS** | Manually Effort | 180 Minutes |
| **TO BE** | Automated Excel to JSON Function | 10 Minutes |
|  | Total Saving/Efficiency | **~170 Minutes** |

* We have 100 and 20 reports around 25 modules. For each,
* Manual effort to validate single report is —Time taken process it is consuming around 180 minutes.
* By using EXCELTOJSON function –Will complete with in 10 minutes.
* Overall time saving is 170 minutes.

## Coding details and explanation with steps:



Basically, it does the following:

1. Set the loading property of the current step
2. Get the file contents using a **FileReader**
   * Once above is completed then step 3 will be executed after the file contents are read
3. Get the File Reader results and convert it to an **XLSX.WorkBook**
   * If the user uploads an empty **workbook** then unset the loading property and return
4. Get the first worksheet of the current workbook
5. Try to convert the worksheet to an array of IRequestItem using the **XLSX.utils.sheet\_to\_json<T>** method
6. Converts A1 range to 0-indexed form.
   * **decode\_range**(range: string): Range;
7. Converts 0-indexed range to A1 form.

* **encode\_range**(s: CellAddress, e: CellAddress): string;
* **encode\_range**(r: Range): string;

1. Converts a worksheet object to an array of JSON objects.

* **sheet\_to\_json**<T>(worksheet: WorkSheet, opts?: Sheet2JSONOpts): T[];
* **sheet\_to\_json**(worksheet: WorkSheet, opts?: Sheet2JSONOpts): any[][];
* **sheet\_to\_json(**worksheet: WorkSheet, opts?: Sheet2JSONOpts): any[];

1. Returns an array of values of the enumerable properties of an object. **@param o** **Object** that contains the properties and methods. This can be an object that has been created or an existing Document Object Model (DOM) object.
   * **values**<T>(o: { [s: string]: T } | ArrayLike<T>): T[];
2. Combines two or more arrays. This method returns a new array without modifying any existing arrays.

**@param items** Additional arrays and/or items to add to the end of the array.

* + **push**(...items: T[]): number;

1. Synchronously writes data to a file, replacing the file if it already exists.

**@param path** A path to a file. If a URL is provided, it must use the `file:` protocol. If a file descriptor is provided, the underlying file will \_not\_ be closed automatically. **@param data** The data to write. If something other than a Buffer or Uint8Array is provided, the value is coerced to a string.

**@param options** Either the encoding for the file, or an object optionally specifying the encoding, file mode, and flag.

If `encoding` is not supplied, the default of `'utf8'` is used.

If `mode` is not supplied, the default of `0o666` is used. If `mode` is a string, it is parsed as an octal integer. If `flag` is not supplied, the default of `'w'` is used.

* + export function **writeFileSync**(path: PathLike | number, data: string | NodeJS.ArrayBufferView, options?: WriteFileOptions): void;

1. Determines whether {path} is an absolute path. An absolute path will always resolve to the same location, regardless of the working directory. @param path path to test.
   * **resolve**(...pathSegments: string[]): string;
2. Converts a JavaScript value to a JavaScript Object Notation (JSON) string.

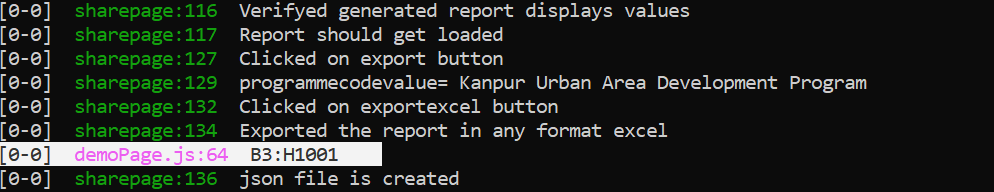
**@param value** A JavaScript value, usually an object or array, to be converted. **@param** replacer A function that transforms the results.

**@param space** Adds indentation, white space, and line break characters to the return-value JSON text to make it easier to read.

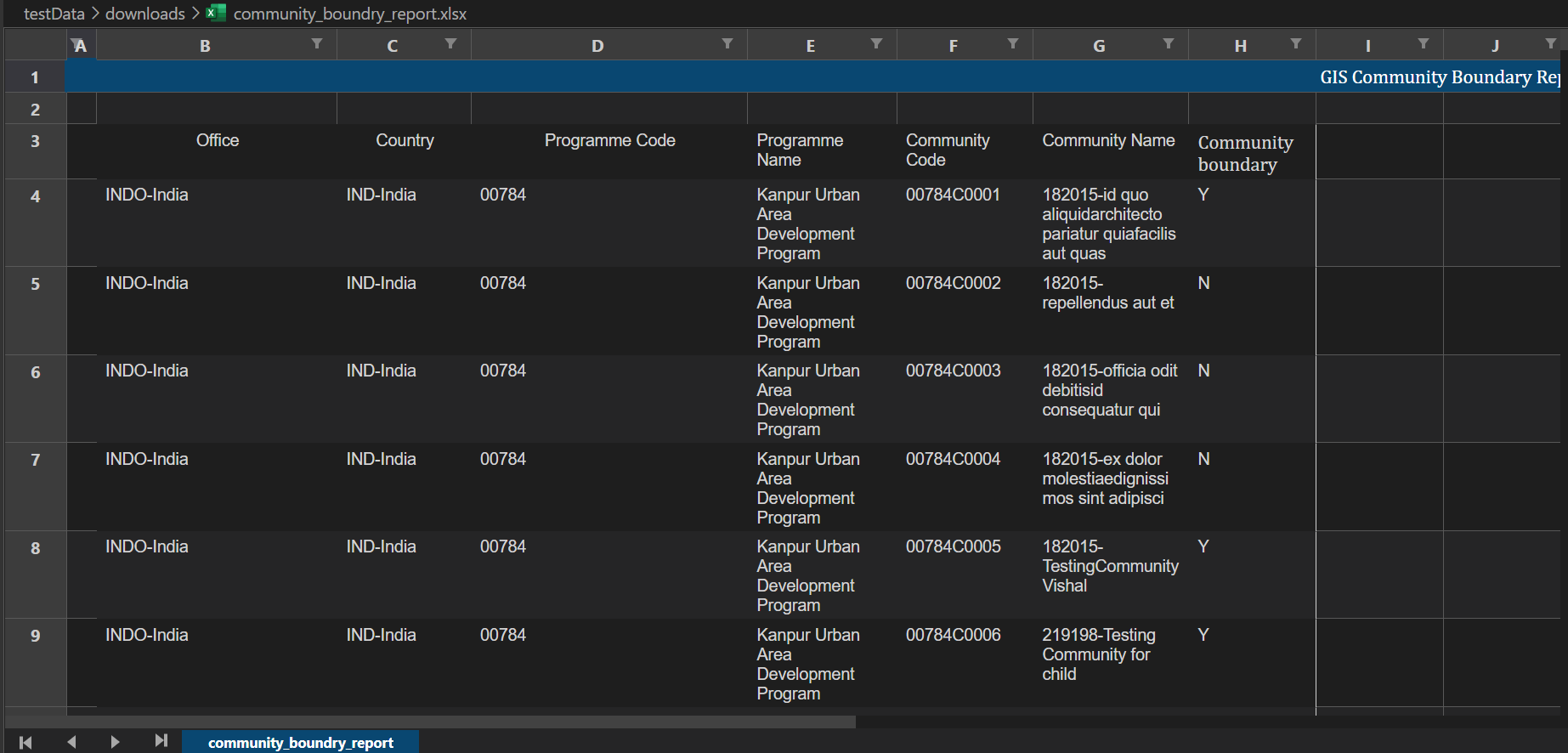
* + **stringify**(value: any, replacer?: (this: any, key: string, value: any) => any, space?: string | number): string;

## Execution status

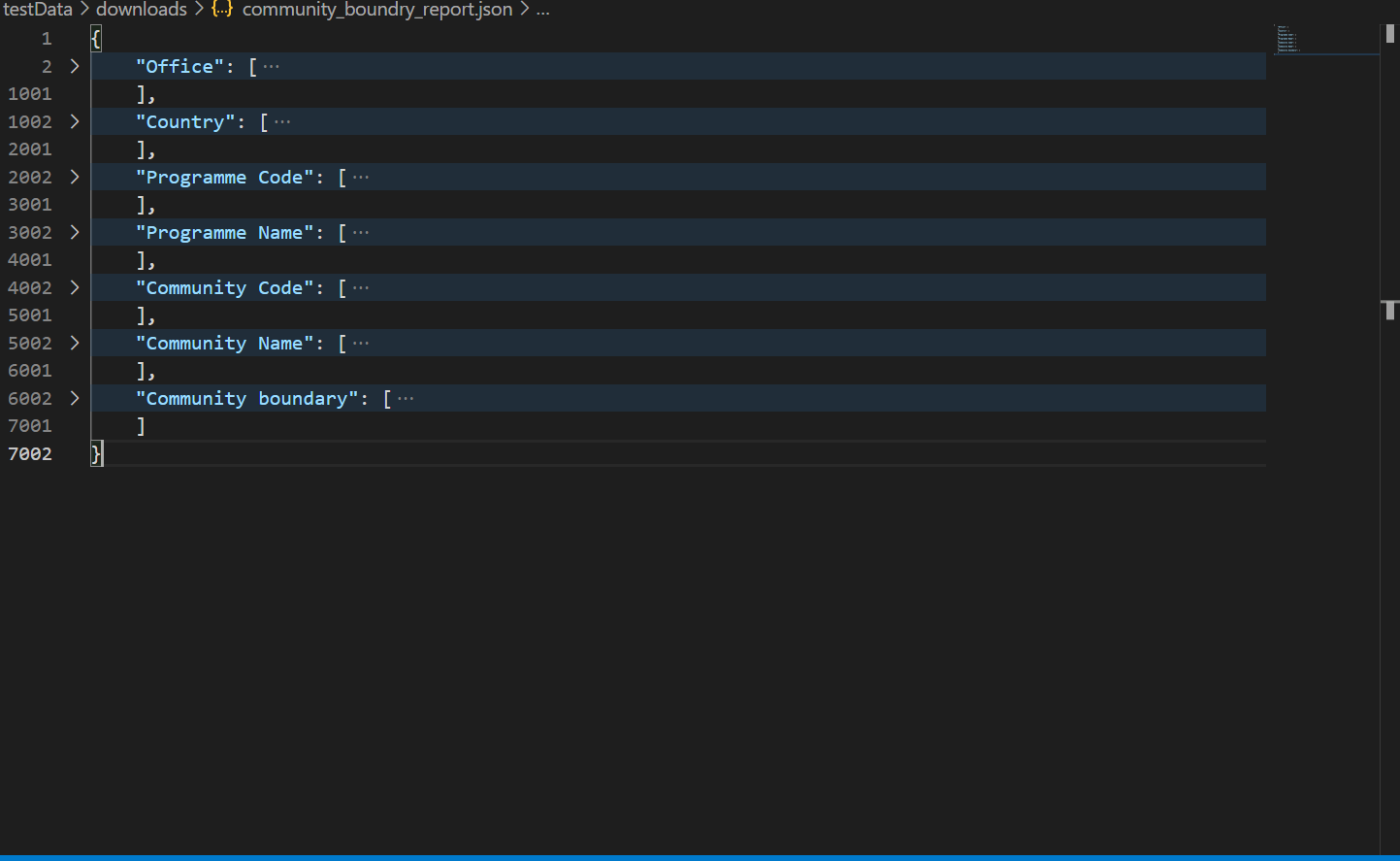
### JSON file is created



### Downloaded Excel File



### Newly generated JSON file from above .xlsx file



### Final Output (JSON file created and comparison result) after executing the Script

