ps-plugin-ExcelTools

Overview

Provides custom smart services and functions for working with Excel and CSV files:

Smart Services

- Import CSV to Database
- Import Excel to Database
- Merge Excel Documents
- Convert HTML to CSV
- Convert Excel to CSV
- Encrypt Excel with Password

Functions

- Read Excel Sheet Paging
- Read Excel Cell by Name
- Read Excel Cell by Number
- Query Appian Logs
- Export Datasubset to Base64

NOTE: If migrating to the Excel Tools plugin for the first time and using the (now deprecated) getdatasubsetdownloadlinkfromprocess function, please delete all old versions of the ExportableDataSubset CDT from the environment AND update your model to point to the new version {urn:com:appian:ps:excel:types}ExportableDataSubset

For exporting data to CSV and Excel, use out of the box Appian smart services.

2.2.13 - RC Running Release Notes Functionality Updates:

- REMOVED Export servlets
- Added printing cell string on cells of type FORMULA in readexcelsheetpaging
- Added an explicit commit in Export SQL to Flat File to avoid a defect in MySQL and MariaDB
- Improved MergeExcelDocuments memory overhead

Smart Services

Import CSV to Database

The Import CSV to Database Smart Service is used to import a CSV (with some columns optionally encrypted.) to a table in a database. Column names that are specified as encrypted will be decrypted during the import. Note: The 'Encryption Service', 'Allow' checkbox needs to be checked when supplying encrypted CSV's.

1. Accepted format for "datetime" is: 2019-12-22T12:24:30Z
2. Accepted format for "date" is : 2019-12-22
3. Accepted format for "time" is 17:20:05

Data Tab

Input	Data Type	Required	Multiple	Description
Additional Columns	Text	No	No	Text string of additional columns separated by comma to be populated for each parsed record
Additional Values	Text	No	No	Text string of additional values separated by comma to be populated for each parsed record
CSV Doc	Document	Yes	No	The CSV document to parse

Input	Data Type	Required	Multiple	Description
CSV Header Names	Text	No	Yes	A list that contains the header names in the CSV to be imported into the database. The length of this list must match the length of the Database Field Names list. Columns that exist in the CSV but are not included in this list will be omitted from being written to the database.
Column Names	Text	No	No	The comma separated column names if 'File Has Header' is set to false
Data Source Name	Text	No	No	The Data Source Name - e.g. jdbc/BusinessDS. This is the value you can grab from *admin console --> Data sources --> NAME OF YOUR DATA SOURCE*
Database Field Names	Text	No	Yes	A list that contains the names of the database fields that the data will be inserted into. The length of this list must match the length of the CSV Header Names list.
Delimiter	Text	Yes	-	The CSV delimiter
Charset	Text	No	-	The encoding for the input csv, default: UTF-8
Encrypted Column Names	Text	No	No	The comma separated column names of the encrypted columns that will be decrypted
File Has Header	Boolean	Yes	-	If this is set to false then 'Column Names' must be provided.
Table Name	Text	Yes	-	The destination table name where the data will be imported.

Output	Data Type	Multiple	Description
Error Message	Text	No	The error message if an error occurred
Success	Boolean	No	Set to true if success and false if an error occurred

Import Excel to Database

This smart service is used to import an excel file to a table in the database. Please note that

1. Correct format for datetime is: 2019-12-22 12:24:30
2. Accepted Date format is : 2019-12-22

Data Tab

Input	Data Type	Required	Multiple	Description
Batch Size	Integer	No	No	How many rows, at max, at a given time should be inserted
Data Source Name	Text	No	No	The Data Source Name - e.g. jdbc/BusinessDS. This is the value you can grab from *admin console --> Data sources --> NAME OF YOUR DATA SOURCE*
Database Field Names	Text	No	Yes	A list that contains the names of the database fields that the data will be inserted into. The length of this list must match the length of the Excel Header Names list.
Excel Document	Document	Yes	No	The excel file
Excel Header Names	Text	No	Yes	A list that contains the names of the excel table header fields to be imported into the database. This allows for the ingestion of table data even when the table data is not first bit of data on the excel sheet. The length of this list must match the length of the Database Field Names list. Columns that exist in the excel data table but are not included in this list will be omitted from being written to the database.
New Columns	Text	No	No	The comma separated column names of the new columns that will be decrypted. Value should be something similar to *column1,column2* and not *"column1,column2"*
New Column values	Text	No	No	Comma seperated values to be inserted for each entry in your excel file.
Number of columns to read in a row	Integer	No	No	How many columns, starting from the 1st column should be read from the Excel
Sheet Number	Integer	No	No	Which sheet from the excel should be read
Table Name	Text	Yes	No	The destination table name where the data will be imported.

The output tab is as follows

Output	Data Type	Multiple	Description
Error Message	Text	No	The error message if an error occurred
Success	Boolean	No	Set to true if success and false if an error occurred

Merge Excel Documents

Documentation currently not provided. Please see smart service configuration.

Convert HTML to CSV

- Converts a single table within an html document into a csv file
- Converted document will be created with the same name as the original document
- Converted document will be created in the same folder as the original document

Data Tab

Input	Data Type	Required	Multiple	Description
HTML Document	Document	Yes	No	The Excel document to parse
CSS Query	String	No	No	The css query to identify the table within the document to parse, e.g. "table", "#tableId", or ".tableClass"

Output	Data Type	Multiple	Description
Return Document	Document	No	The converted csv document
Error Occurred	Boolean	No	Whether an error occurred
Error Text	Text	No	The error message

Convert Excel to CSV

Documentation currently not provided. Please see smart service configuration.

Load CSV to Database

The "Load CSV to Database" smart service is designed for use with an on-premise (not Appian Cloud) MySQL DB only and uses the native LOAD CSV functionality from MySQL. Documentation currently not provided. Please see smart service configuration. Due to the specific pre-requisites for using the smart service, the "Import CSV to Database" smart service node is more commonly used.

Encrypt Excel with Password

- Creates encrypted excel workbook in the provided folder and with the provided filename.

Data Tab

Input	Data Type	Required	Multiple	Description
Document To Be Protected	Document	Yes	No	The Excel document that will be encrypted
New Document Name	Text	Yes	No	Name of the encrypted file
Password	Text	Yes	No	Password to encrypt the file
Save in Folder	Folder	Yes	No	Destination folder of the encrypted file

Output	Data Type	Multiple	Description
Document with Password	Document	No	Encrypted Document

Output	Data Type	Multiple	Description
Error Occurred	Boolean	No	Whether an error occurred
Error Text	Text	No	The error message

Functions

readexcelsheetpaging() Custom Function

This Custom Function allows to read an Excel sheet in a paged way. It returns a DataSubset so it can be used for example in a gridField, every row of the returned DataSubset is a Dictionary with only one field: "values" which contains the list of string values of all cells in a row. Now the user's Locale taken into account when reading Cell values. Formula cells with string outputs will print the formatted value. Non string formulas will result in an error.

readexcelsheet

Documentation currently not provided. Please see inline documentation from rule or interface designer.

excelreadcellsbyname

Documentation currently not provided. Please see inline documentation from rule or interface designer.

excelreadcellsbynumber

Documentation currently not provided. Please see inline documentation from rule or interface designer.

queryappianlogs

This Custom Function allows to perform an SQL statement against an Appian CSV log file. Only the following SELECT statement is supported:

SELECT [DISTINCT] [table-alias].column [[AS] alias], ...

FROM table [[AS] table-alias]

WHERE [NOT] condition [AND | OR condition] ...

GROUP BY column ... [HAVING condition ...]

ORDER BY column [ASC | DESC] ...

LIMIT n [OFFSET n] The name of the table must correspond to the name of the CSV log to query. The function is limited to the first 1000 rows.

Input Parameters

Input	Data Type	Required	Multiple	Description
sqlStatement	Text	Yes	No	SELECT SQL Statement to perform
subFolder	Text	No	No	Specify with a value if the CSV log file is in a log subfolder. Accepted values: "audit", "data-metrics", "perflogs"
columnTypes	Text	No	Yes	Ordered list (based on the column order in the SQL Statement) types the column will be formatted. Accepted values: "Int", "Double", "Date", "Time", "Timestamp", "String", "Boolean". If Set it must have the same amount of values as the number of columns in the SQL Statement
timestampFormat	Text	No	No	Specify the format of the timestamp columns in the CSV file. For instance, "d MMM yyyy HH:mm:ss z" is the format of the timestamp in system.csv

Input	Data Type	Required	Multiple	Description
hasHeader	Boolean	No	No	Specify a value if the CSV has a header or not. Default is true which means the header is in the first line of the CSV. If set to false then columns are named sequentially COLUMN1, COLUMN2, ...

Make sure that the order of the input fields is correct. Appian plugin inputs must be passed by position, not keyword syntax. Although some paramters are optional, for passing: *hasHeader: false*, all the previous paramaters must be included as seen below:

```
queryAppianLogs("select * from login-audit",null,{"String", "String", "String","String","String","String"},null,false)
```

Output Parameter

The returned parameter is a CDT made of the following fields:

{

success: Boolean value, based on the success of the SQL Statement execution

errorMessage: Text value, describing an SQL error (if any) returned during the SQL Statement execution

data: List of CDT, where each field corresponds to the column name (or alias assigned). Field type corresponds to the ordered values > set in che input "columnTypes", or Text for all of them otherwise

}

validatedocumentheaders

Performs validation on a provided CSV or XLSX documents headers. Validation indicates if the provided expectedHeaders exist in the provided document.

Input Parameters

Input	Data Type	Required	Multiple	Description
document	Document	Yes	No	CSV or XLSX document to validate
expectedHeaders	Text	Yes	Yes	List of header names that you expect to be in the document
sheetNumber	Number(Integer)	No	No	The sheet number in the XLSX you would like verified. Defaults to 0 (sheets are zero indexed) if XLSX file is provided

Output Parameter

Dictionary is returned in the following format:

{

success: Boolean value, based on the success of the validation

errorMsg: Text value, describing the error that occurred

matchedHeaders: List of Text, indicating which expectedHeaders were found in the document

headersNotMatched: List of Text, indicating which expectedHeaders were not found in the document

additionalHeaders: List of Text, indicating which headers were found in the document but were not in expectedHeaders

numberOfMatchedHeaders: Number(Decimal) value, indicating the number of expectedHeaders that were found in the document

percentageMatched: Number(Decimal) value, indicating (matchedHeaders / expectedHeaders) x 100

}

exportdatasubsettobase64

This custom function generates a base64 text string representation of an Excel spreadsheet populated with the datasubset.

This expression must only be used within an Appian webApi in the context of exporting a report to Excel. Using this function in a process model will result in a large amount of text being stored within the Appian engines memory and could cause scalability problems.

Refer to the application "DSEW Datasubset Export Webapi" available as an attachment in the Excel Tools shared component for an example on how to use this function:

- DSEW_testExportWithFiltersAndSort: User interface showing how to invoke the export and pass in parameters to configure the report data source entity and filters/sorts. The parameters are sent to an Appian webApi as an encoded JSON string.
- DSEW_exportToExcel: Appian webApi showing how to interpret the JSON string parameter, execute the query entity and generating the base64 text string representing the Excel spreadsheet. The webApi also contains HTML/JS code to open the spreadsheet in the user browser.

Deprecated

Export Sql to Excel Smart Service

This takes an SQL statement and popluates the data into the excel workbook. You can specify which tab to write the data to. Tabs are zero based index with first tab =0. You need to have a JNDI defined and the value of the JNDI needs to be passed to this plugin.

In addition, you can write data to any arbitrary cell by defining an array of cell numbers e.g. {"A1","B6","Al4"} and passing values to these cells. This is a one-to-one cell data population.

Currently tested on MySql and Oracle database

Known Issues: If you are using this with Oracle where the columns are created with lower case and require double quotes to be appended to the SQL, it doesnt work.

Export Report Servlet

A servlet that can be used inside an a!safeLink to simulate export to excel of Tempo reports that rely on getportalreportdatasubset().

See also: https://home.appian.com/suite/tempo/news/entry/f-109980

Tips

You need multiple of these nodes to write to different sheets/tabs of the excel report. The excel report template MUST have these sheets defined otherwise the node will fail

You can configure graphical chart on an excel report for a defined set of rows/columns and when the data is populated in these cells, the data should automatically get translated into the graph (as per your configuration on the template)

Use multiple of these to combine Appian and RDBMS data where required. For e.g. You can write the data from Appian report followed by SQL data in the next tab.

Example

Cell_Keys : Array of cell keys where info needs to be written to e.g: {"A1","B1","C1"}

Cell_values Array of cell values where info provided will be written (for the corresponding values in Cell_Keys) e.g: {"Apple",100,"10/Oct/2012"}

Document_name_to_create Name of the document to be generated.

Document_save_directory Folder to save the generated document

Document_to_overwrite If a document generated should overwrite another existing document, use this. Useful if you are invoking this to write to different sheets of an excel document.

Excel_base_template The excel template on which the output will be written to

include_header_row Boolean true() or false() allowing header row to be written

Jndi Name Name of the JNDI (Quick help: Look at your database node to get the JNDI name.) This will resemble java:/jdbc/AppianDS

Sheet_number Excel workbook Sheet number to write to. First sheet is 0. If your excel has only one sheet, this should be 0 (required)

SQL SQL to use or query. You can use an SQL that contains Select * here. Do not use Stored Procedures with output parameters with the exception of Oracle since it MUST have an output parameter. For this case, the Oracle SP should be called with SQL such as call oracleSP(,,?) "?" represents the the output parameter to be used for mapping the sysref cursor that manages the result set.

Starting_cell Cell to start writing the report output. For e.g. if you want the output of sql to start from cell "A4" onwards, specify "A4" here.

Export SQL to Flat File Smart Service

Delimiter The separator between data fields in the file.

Document_name_to_create Name of the document to be generated.

Document_save_directory Folder to save the generated document

Document_to_overwrite If a document generated should overwrite another existing document, use this. Useful if you are invoking this to write to different sheets of an excel document.

End_of_line The character to be used as a line break. For a carriage return use char(13), for line feed use char(10) and for CR/LF use char(13)&char(10).

Extension The file extension of the file generated.

include_header_row Boolean true() or false() allowing header row to be written

Jndi Name Name of the JNDI (Quick help: Look at your database node to get the JNDI name.) This will resemble java:/jdbc/AppianDS

SQL SQL to use or query. You can use an SQL that contains Select * or Stored Procedures. Stored procedure format differs based on RDBMS.

- MySQL/MSSQL - Your stored procedure should not have output parameters, and should be in lie format ="{call My_SP(<input_val1>,<input_val2>,...,<input_valN>)}"
- Oracle - requires an output sysref cursor to bring back a result set. In your SQL statement, place a ? (question mark) character in the correct position of the parameter as defined in the SP. For example, ="{call My_SP(<input_val1>,<input_val2>,?)}" would mean that you have an SP with 3 parameters - 2 INs and 1 OUT representing the sysref cursor.

*See the attached Resources application and SQL files for real example uses.

Updating Project

Setup Maven

Download and configure Apache Maven

Setup Eclipse

1. Generate new Eclipse project files by running `mvn eclipse:clean eclipse:eclipse` from the root project folder
2. Open Eclipse
3. File > Import > Existing Project Into Workspace
4. Under `Root Directory` choose the location you copied this repo to
5. Click Finish
6. Configure build path to include all libraries under `built-tools\lib\appian\18.4.0\_admin\deployment\web.war\WEB-INF\lib`

Building

Development Build

To create a plug-in jar for testing, run the following Maven command:

- `mvn clean package`
- The jar can be found in `/target`

Release Build

To create a new public release, run the following Maven commands:

- `mvn release:clean`
- `mvn release:prepare -DautoVersionSubmodules=true -DpushChanges=false`
- `git push origin master --tags`
- The release jar can be found in `/target`

Previous Release Notes

2.4.0 Release Notes Functionality Updates:

- Import CSV to Database
  - Added the ability to map CSV header values to database field names
- Added new function performdocumentvalidation

2.3.0 Release Notes Functionality Updates:

- Import Excel to Database
  - Added the ability to map Excel header values to database field names

2.2.17 Release Notes Functionality Updates:

- Changed dependency commons-compress from 1.20 to 1.21
- Changed dependency org.jsoup from 1.13.1 to 1.14.3

2.2.12 Release Notes Functionality Updates:

- Added new function readexcelsheetpaging
- Deprecated readexcelsheet

2.2.10 Release Notes Functionality Updates:

- Replacing deprecated APIs
- The "Import CSV to DB" smart service supports adding static columns and static values to the records provided in the CSV file
- The "Import CSV to DB" smart service does not fail if header/column list does not match the table structure. Only columns that match the table will be processed. Empty column data in CSV will result in NULL values in the corresponding fields in the table.
- The "Import CSV to DB" smart service supports datetime format in the "ISO-8601" standard
- The "Load CSV To Database" smart service has been deprecated

Documentation updates:

- For "Import CSV to DB" acceptable datetime format has been updated
- "Setup Eclipse" section updated with instructions for configuration of build path
- "Previous Release Notes" section created at the bottom of this document and previous release notes moved there
- Other minor documentation updates

2.2.9 Release Notes Functionality Updates:

- The “Import CSV to DB” smart service updated to roll back the transactions when an exception is thrown

2.2.8 Release Notes Functionality Updates:

- Fix defects with missing full file path structure when exporting data

2.2.7 Release Notes Funcionality Updates:

- The "Import Excel to DB" smart service no longer errors if no value is passed for the inputs "New Columns" and "New Columns Values"
- The "Import CSV to DB" smart service now returns a readable error message if there are extra blank columns in the CSV file
- The "Query Appian Logs" function no longer skips the first row of data is the input “hasHeader” is passed as false
- The "Convert Excel To CSV" smart service has been updated to utilize the inputs "numberOfColumnsToReadInRow" and "rowNumberToReadFrom"