SAS© Visual Analytics Table Editor

Technical and User Manual

Tabulator4VA – v7.0

João Oliveira – joao.oliveira@sas.com

SAS South Western Europe – Customer Advisory

Doc v26 – 11OCT2021

Revision Sheet

Release No.	Date	Revision Description
Rev. 1	20JUN2020	**Initial Work**
Rev. 2	12AUG2020	Installing the Job Execution and registering it on the parameters
Rev. 3	11SEP2020	Additional functionalities, setup and usage. Requires tabulator v4.8 +
Rev. 4	25SEP2020	Added 'Paste from Clipboard' functionality.
Rev. 5	09OCT2020	Added the Control File managing visible columns and its validations
Rev. 6	16OCT2020	Added the ability to export the App logger as JSON, HTML or CSV
Rev. 7	23OCT2020	Added on the Control File a column default value fallback on the validation object
Rev. 8	05NOV2020	Added display #Rows and select cell value from a Drop-Down List
Rev. 9	11NOV2020	Added Header Filtering mapping the Drop-Down List criteria
Rev. 10	13NOV2020	Added controlling 'validUsers' also by User Group.
Rev. 11	10DEC2020	**Version 4.0** – Includes multiple new features: - Three ways how Control Files can be read - Table Validation on (re)Loading - Default behaviour if no Control File(s)/definitions found - Tracking All Changes made - Rearranged Cell Context Menu o Adding Rows - Defaulted row - Three Saving modes to handle Save in Memory and on Disk And performance improvements, specially needed on validating an entire table (is made cell per cell) and on receiving data from the Clipboard.
Rev. 12	11JAN2021	Possibility to associate a formula to a column, which can be used to validate its value. Added the ability of a cell receiving the value, which can be calculated by a formula, from another cell or from a lookup table. New functions were added: ***year, month, week, day, time, hours, minutes, seconds, lookup***
Rev. 13	19JAN2021	The ability to a cell receiving its value from another cell: • can now include multiple columns • if formula, that is also applied on column mass change
Rev. 14	20FEB2021	Introducing the ability to use formulas on strings (list and syntax as appendix on this document) Introducing the monitoring/inspecting invalid cells. Invalid Cells tooltip will show the violations (see explicative video) and it's also possible to export / print as well.
Rev. 15	08MAR2021	- On the invalid cells' information, a suggested value is presented. - Introducing aliases for the date and time functions, like 'getTime' ⇔ 'time' (see manual); additional functions 'getReportUserId', 'getReportUserName', 'getReportUID', 'getReportSessionUID'. - Adding on the Context Menu, the ability to call external Services. Those can be called in 'background' or 'foreground' (new Window) mode.
Rev. 16	26MAR2021	- Ability to define column aggregations, either to be shown on the top and / or at the bottom. Functions possible to use are the most common statistical functions like, ***sum, avg, mean, mode, median, ...***

Rev. 17	15APR2021	**This release – v4.5**, introduces: 1) recovery of unsaved changes in case of closing the report. 2) Independent Header filtering (not dependent from the cell editorSelect config), in numeric and string/alphanumeric columns. 3) ability to freeze columns, either on the left and/or right of the table. 4) Informative messages colouring (at the bottom, underneath the buttons).
Rev. 18	22APR2021	- Defining ***validUsers*** at the ***.meta.ctrl.json*** level to add one additional security level. - Ability to configure and then execute in the background External WebServices (e.g.: SAS JobExec) at the Save moments - External Services calls / executions, are logged into the Table Tracker - Saving the App log on the file system on ./Logs upon closing report. - If a cell content has changed based on the formula, that is presented as cell tooltip and if user clicks back on the cell for another edition, what's presented is the used formula a not its result.
Rev. 19	12MAY2021	**Version 5.0** – Includes multiple new features: - **Multi-User : User Concurrence on Save** o VATableEditor will check if table has been modified and if those modifications were made by another VATableEditor User or not. In case that was another VATableEditor User, those changes are verified to assess if have any impact on the changes about to be saved… - **Security**: o align/sync with Viya Group/User Table Authorisations o implement own authorisation schema per table - Improved the Auto-loading (if not loaded) of tables used on 'lookup' and/or as sources for 'editorSelect' from 'table' - Redesign of Message Boxes to reflect the overall look - New functions to give more integration options on calling SAS Services and/or 3rd party webservices. Also, some bug fixing and performance improvements.
Rev. 20	04JUN2021	**Version 5.5** – Changes on the External Services Definitions, Capturing a (tweet like) comment – The reason – for the changes, at saving moment, 'editorSelect' "dynList" as source for a cell value obtained from a LoV and, more aggregation functions: ***sterr*** (aka stdErr/stderr), ***gMean*** (Geometric Mean), ***hMean*** (Harmonic Mean) and ***zScore***. - External Services are now defined at Report Level, which allows to reuse those in multiple tables (avoids repeating the service definitions). The JSON object schema maintains as before, an at table level – for both possibilities, Context Menu call and/or at Saving Moment, it's just naming to the Service Name defined on the Report Level. - Tweet Like comment for capturing the reason for the changes, this is subject to a parameter following the hierarchy, HTML is overwritten by **.meta.crtl.json**, which is overwritten by the **report .crtl.json**, which is overwritten by the definition at table level.

		- Dynamic Drop-Down list of values for 'editorSelect', meaning that in the same column a cell might have a different LoV than other cell, this based on another cell via using 'lookup' function.
Rev. 21	18JUN2021	**Version 5.6** – Changes on the External Services Definitions organizing the after-execution actions, into a group "afterExecActions". Introduces the ability to define table initial sort and how the rendering mode.
Rev. 22	02JUL2021	**Version 5.7** – Introduction of new functions to allow more possibilities for integration with, either SAS Viya Applications and/or 3rd party.
Rev. 23	14JUL2021	**Version 5.8** – Introduction of actions on the 'gotoPage' action "afterExecActions". On numeric cell validation it's possible to set a formula for the 'min' / 'max', e.g. *"min": "getColumnValue(<column_name>)"*
Rev. 24	23JUL2021	**Version 6.0 –** Total integration with Visual Analytics modes – ***Editing* | *Viewing*** – in the way that: ➢ New data sources and other #VATableEditor objects can be introduced on the report and, ➢ Adding / removing / moving columns on the new and/or on an existing #VATableEditor object ➢ Once in 'Editing' mode, several functionalities, like ***Table Validation***, aren't performed to ease the report design. Those are automatically reactivated once the report gets into ***Viewing*** mode. **All this without the need to reopen the report!**
Rev. 25	31AUG2021	**Version 6.5 –** Adding new functionalities like: - **Security**: Ability to define, on the main/master config file, Admin group(s) and/or user(s). This will filter the Export Menu, etc. - **Header filtering functions**: ***in, !in, btw, !btw*** *(equivalent to* ***out****)*, **<!>.** See below the detailed description and how to use. - **Cell presentation mode**: It's now possible to present **numeric cells** as: o **Check Box:** (optionally presenting only the 'tick', or the 'cross' or both). This ONLY works with Boolean type of numeric column containing values **0 | 1**. o **ProgressBar**: it's possible to provide a set of colours, max. etc. o **TrafficLight**: identical to ProgressBar o **Rating Stars**: possibility to provide the max number of stars
Rev. 26	11OCT2021	**Version 7.0 –** Introducing Global External Services (available to all reports, ability to apply a formula from a cell to the column …– - **External Services:** Ability to define on the main/master config file, ***External Services***. These have the same properties as the defined on the Report Control file. <u>In case of a External Service being defined in both, the one in the Report control file wins</u>. - **Changing Column value:** If the cell value was / is calculated by a formula, is now possible to apply, either the cell value or cell formula. If the cell propagates its calculation to another column, those are also affected. The rollback can be applied in any of the modified columns.

		- **HTML file:** simplification of the definitions. The call to Exit function on the '*window.onunload*' optional. Useful if the client/user wants to run its own code on this event. - **New Parameters:** may be defined on the HTML, main control file and report control file. The deepest wins. ○ **ReopenReportOnTableChanges :** Boolean, default 'false' ○ **OpenInFullScreen:** Boolean, default 'false'

TABLE OF CONTENTS

GENERAL INFORMATION

System Overview

It’s a component developed by SAS South Western Europe Customer Advisory for being used as a Data Driven Content on SAS © Visual Analytics, that allows users to edit the content of a SAS Viya CAS table. It is possible to change the content of individual cells, change the content of an entire column, delete and add rows, saving the changes on the in-memory table as well as making those changes persistent.

It’s a software component developed entirely in javascript on top of Tabulator, a opensource and MIT license based javascript framework, making use of other MIT License components such as JQuery, Moment and FontAwesome.

Authorized Use Permission

SAS© provides you with access to the use of this component, including documentation and other product information (collectively the “Documentation”), under the <partnership contract> and/or the Non-Disclosure Agreement (NDA). As per the scope of those contracts, you have access to SAS download areas, communication forums, and other services (collectively "Services"), software, including developer tools and sample code (collectively “Software”), and Application Program Interface information (“APIs”). The Documentation, Services, Software, and APIs (including any updates, enhancements, new features, and/or the addition of any new Web properties to the Web Site), are subject to the following Terms of Use ("ToU"), unless we have provided those items to you under more specific terms, in which case, those more specific terms will apply to the relevant item.

Points of Contact

Information

The points of contact (PoCs) that may be needed by component user for informational and troubleshooting purposes would be the country SAS Customer Advisory interfacing with the user of this software component.

Help Desk

Help desk information including responsible personnel phone numbers for emergency assistance is currently not available as the software component is provided as is.

The user of this software component has, however, access to its local PoC which then can engage with the relevant SAS people to resolve any eventual situation.

Acronyms and Abbreviations

List of the acronyms and abbreviations used in this document and respective meaning.

App:	Application
VA:	Visual Analytics
DDC :	Data Driven Content
CSS :	Cascade Style Sheet
JS:	Javascript
LIFO:	Last In First Out

INSTALLATION – VIYA 3.5

This section explains what components would be installed on the SAS© Viya Server

Directory Hierarchy

On the SAS Viya Server all the Data Driven Content special code / libraries / html files **must be** installed under: **/opt/sas/viya/home/var/www/html/htmlcommons/** **<u>therefore from this point forward all locations are relative to this location</u>**.

1. When accessing the server change directory to the one mentioned above.
2. Copy the given ZIP file into
3. Change to the directory in **1**
4. unzip the provided ZIP file
5. As result of that you should have the following directory structure under

/opt/sas/viya/home/var/www/html/htmlcommons/DataDriven/

```
./cssjs
./cssjs/js
./cssjs/css
./cssjs/webfonts
./Tabulator4VA
./Tabulator4VA/css
./Tabulator4VA/js
./Tabulator4VA/ReportDefinitions
./Tabulator4VA/Logs
```

Foundation CSS/JS Libraries

Cascading Style Sheets

- **./cssjs/webfonts**
  - fa-*.* > *fontawesome fonts*
- **./cssjs/css**
  - fontawesome-all.min.css > *fontawesome icons*
  - tabulator_*.min.css > *tabulator style sheets*. On the html file you can choose one
  - tabulator_*.min.css.map > *tabulator style sheets map files*
  - jquery-ui.min.css > *jquery user interface base style sheet others on **./jsquery-ui***
- **./cssjs/css/images**
  - ui-icons_*.* > *jquery-ui icons*
- **./cssjs/css/jquery-ui/<*>** > *jquery-ui style sheets, contains the respective ./images folder*

Javascript

- **./cssjs/js**
  - jquery-2.2.3.min.js > *JQuery library*
  - fontawesome-all.min.js > *font awesome library … for the fancy icons*
  - jquery-ui.min.js > *JQuery User Interface library*
  - moment.min.js > *moment library (for handling dates)*
  - moment-with-locales.min.js
  - moment.min.js.map > *moment library map file*
  - math.min.js > *library supporting the use of formulas (mathjs.org)*
  - xlsx.full.min.js > *library supporting export to EXCEL (github.com/sheetjs/)*
  - jspdf.min.js > *libraries supporting export to PDF (github.com/MrRio/jsPDF)*
  - jspdf.plugin.autotable.min.js > *(github.com/simonbengtsson/jsPDF-AutoTable)*

    - tabulator.min.js > *Tabulator library*

Specific CSS/JS Libraries – SAS ©

The bellow enunciated libraries are SAS intellectual property and can be used, by partners and customers, as is.

HTML page (example)

This page is an example although fully functional including all the required CSS and JS. Please check it as **all the includes** are using the relative paths.

- **./Tabulator4VA**
    - DDC_Tabulator4VA_(2).html

Cascading Style Sheets

- **./Tabulator4VA/css**
    - Tabulator4VA_Styles.css

Javascript

- **./ Tabulator4VA/js**
    - Tabulator4VA_Libs-min2.js
    - Tabulator4VA_Main-min.js
    - Tabulator4VA_Core-min.js
    - Tabulator4VA_Modules-min.js
    - Tabulator4VA_Foundation-min.js

Job Execution

The Job Execution, in the current version, is required for Adding rows.

With the installation .ZIP are provided two files:

- **VATableEditor.json**
    - This is a file that installs the job execution via importing it on SAS Environment Manager > Content. See installation details below for the detailed steps.
    - Also creates a folder, **'Examples'**, which contains within the **'Reports'** subfolder is an example using the SASHelp tables 'SNACKS' and 'CARS' (respective csv files are under the subfolder **'Data'**, so either are imported to the **'Public'** *caslib* or there made available creating a copy from the **sashelp**.

Installing the Job Execution program

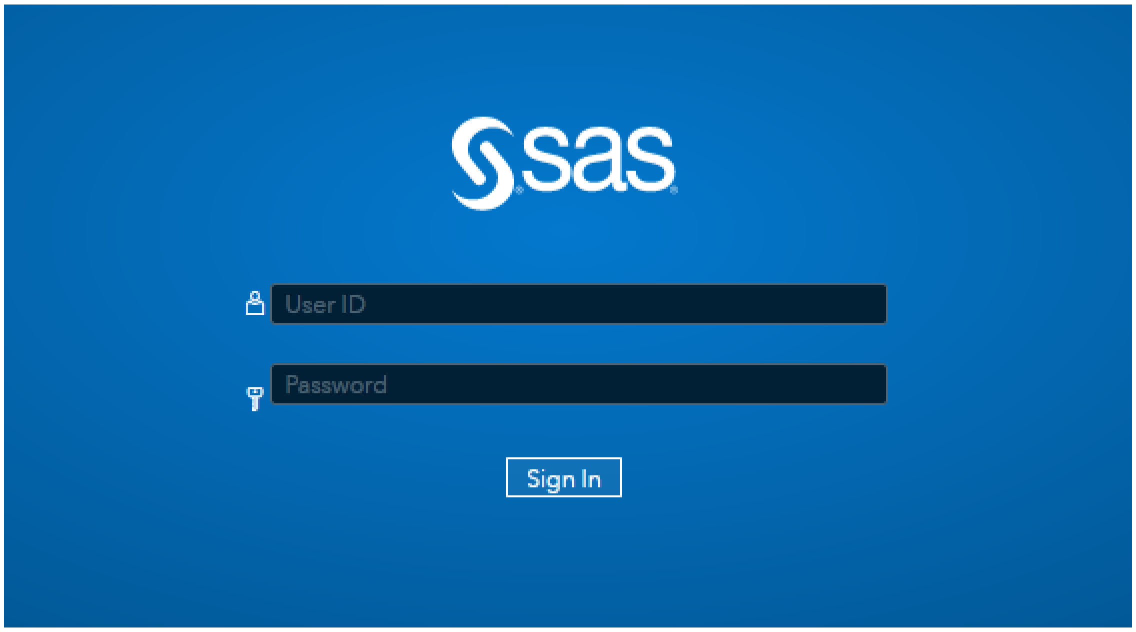
Here is presented / described how to install the SAS program by importing the **VATableEditor.json**

Pre-Requisites

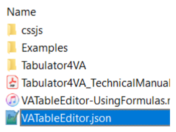
A user ID and password is required to log onto the application web interface. **For installing the Job Execution it's required Admin privileges.**

VA-TableEditor component was tested on SAS Viya 3.5 and SAS VA 8.5

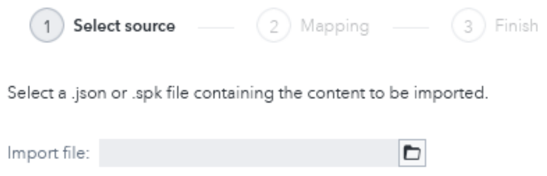
https://<YourServer:port>/SASVisualAnalytics/



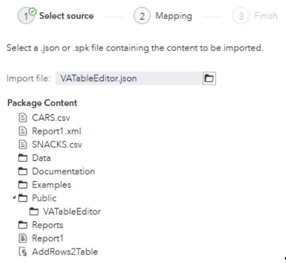
1. Login using an account with Admin grants.



2. Extract the file **VATableEditor.json** from the installing ZIP storing it in your local computer.
3. On the top left corner of the screen, click on the icon and choose **'Manage Environment'** under the group **'ADMINISTRATION'**
4. On the left-hand vertical list of icons, click on **'Content'**
5. On the presented page, on the top-right of the right section, click on the icon **'Import'**


1 Select source
2 Mapping
3 Finish
Select a .json or .spk file containing the content to be imported.
Import file:

6. Click on the field 'Import File' on this area of the presented window.

7. Once you selected the file **VATableEditor.json** from where you stored in your local computer, you will see 

   1 Select source — 2 Mapping — 3 Finish

   Select a .json or .spk file containing the content to be imported.

   Import file: VATableEditor.json

   Package Content
   - CARS.csv
   - Report1.xml
   - SNACKS.csv
   - Data
   - Documentation
   - Examples
   - Public
     - VATableEditor
   - Reports
   - Report1
   - AddRows2Table

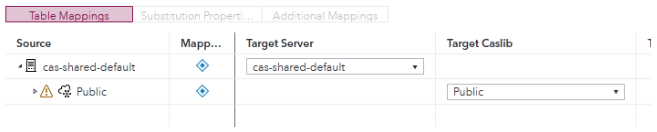
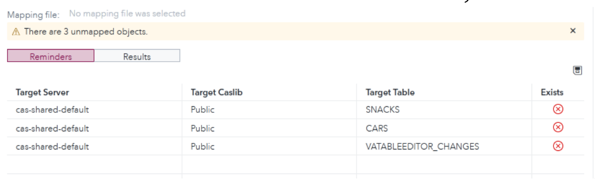
   , so click on '**② Mapping**', which should present a page like 

   Table Mappings | Substitution Properti… | Additional Mappings

Source	Mapp…	Target Server	Target Caslib
cas-shared-default	◈	cas-shared-default	
Public	◈		Public

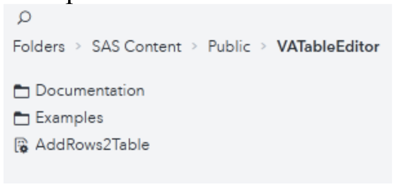
   , then on the '**③ Finish**', will present three inexistent tables 

   Mapping file: No mapping file was selected

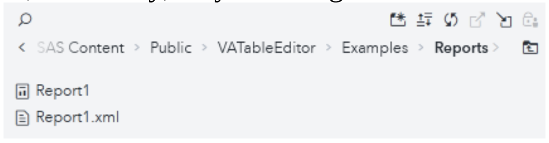
   There are 3 unmapped objects.

   Reminders | Results

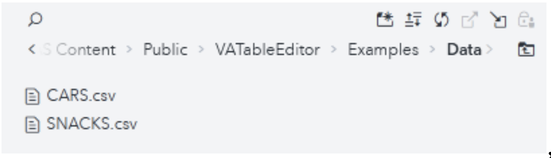
Target Server	Target Caslib	Target Table	Exists
cas-shared-default	Public	SNACKS	⊗
cas-shared-default	Public	CARS	⊗
cas-shared-default	Public	VATABLEEDITOR_CHANGES	⊗

   , which is normal as those do not exist yet (SNACKS and CARS should be either imported or made available from **sashelp** library.. the VATABLEEDITOR_CHANGES will be automatically created upon the first SAVE using of the App, ***so, it's normal get the error saying that table isn't found when the report is open***), then followed **by 'Import'** button.
8. Wait until completion … mind the location on SAS Content (path) where the **AddRows2Table** was installed, 

   Folders > SAS Content > Public > **VATableEditor**
   - Documentation
   - Examples
   - AddRows2Table

   as you, eventually, may need to go there.
9. The report provided as an example is located at 

   < SAS Content > Public > VATableEditor > Examples > **Reports** >
   - Report1
   - Report1.xml

   and CSV tables SNACKS and CARS are located at 

   < S Content > Public > VATableEditor > Examples > **Data** >
   - CARS.csv
   - SNACKS.csv

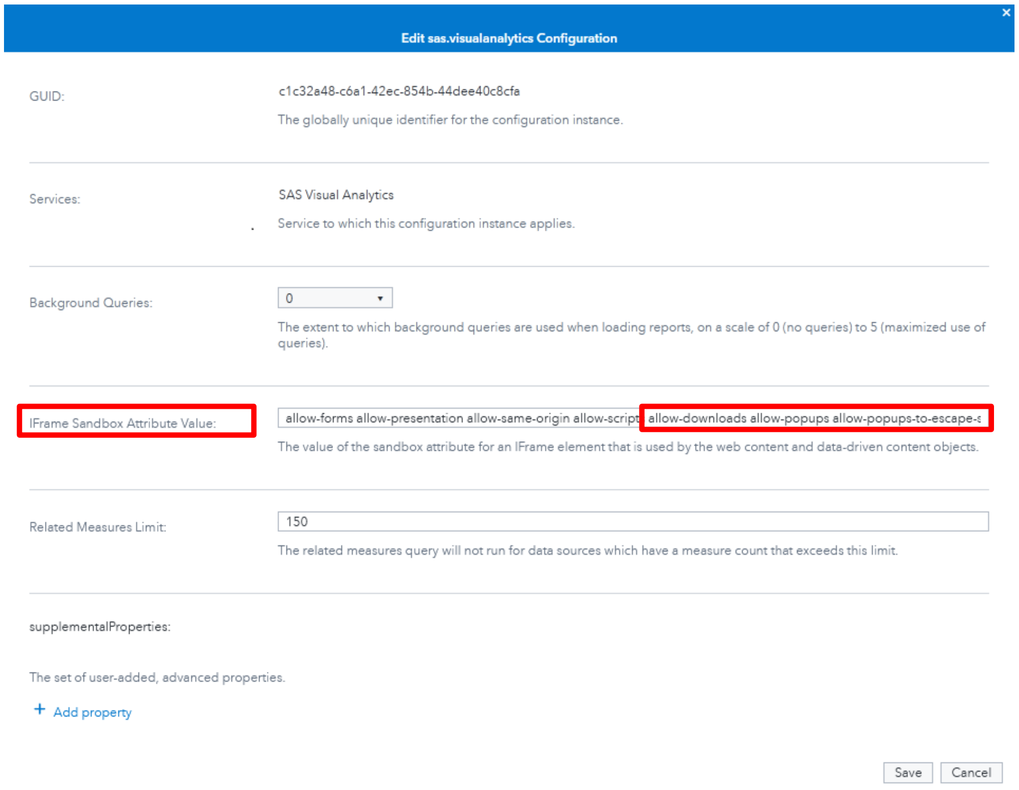
   , so you can import to the '**Public**' library.

With version 4.5, <u>CARS and SNACKS files are also provided as SAS files so the column types are all good</u>. Thus, the preference is to import these instead of the CSV. When importing these files, **you must add a primary key column, UNIQUE_ID, as suggested by the import feature!!**

This said, you should import the table 'SNACKS_LOOKUP.csv' (present also on the ZIP) as the report is using it.

Configuring SAS VA iFrame parameters

This step is required as browsers based on Google's Chromium framework block the possibility to download files and/or printing, and/or opening popups from HTML document **iFrame**. Thus, following the following this note https://support.sas.com/kb/65/978.html (referred on https://www.yammer.com/sas.com/#/Threads/show?threadId=725741941063680) the VA configuration needs to be changed adding the following:
allow-downloads allow-popups allow-popups-to-escape-sandbox to the **'IFrame Sandbox Attribute Value'**. **To do this you should have Administrator grants!**


Edit sas.visualanalytics Configuration

GUID: c1c32a48-c6a1-42ec-854b-44dee40c8cfa
The globally unique identifier for the configuration instance.

Services: SAS Visual Analytics
Service to which this configuration instance applies.

Background Queries: 0
The extent to which background queries are used when loading reports, on a scale of 0 (no queries) to 5 (maximized use of queries).

IFrame Sandbox Attribute Value: allow-forms allow-presentation allow-same-origin allow-script allow-downloads allow-popups allow-popups-to-escape-s
The value of the sandbox attribute for an IFrame element that is used by the web content and data-driven content objects.

Related Measures Limit: 150
The related measures query will not run for data sources which have a measure count that exceeds this limit.

supplementalProperties:

The set of user-added, advanced properties.

SETTING UP THE DATA DRIVEN CONTENT OBJECT

GETTING STARTED

This section provides a general walkthrough on how to setup SAS© Visual Analytics Data Driven Content in order to use this software component.

It's assumed that the user has the necessary knowledge about SAS© VA and therefore about DDC

Pre-Requisites

A user ID and password is required to log onto the application web interface.

VA-TableEditor component was tested on SAS Viya 3.5 and SAS VA 8.5

https://<YourServer:port>/SASVisualAnalytics/

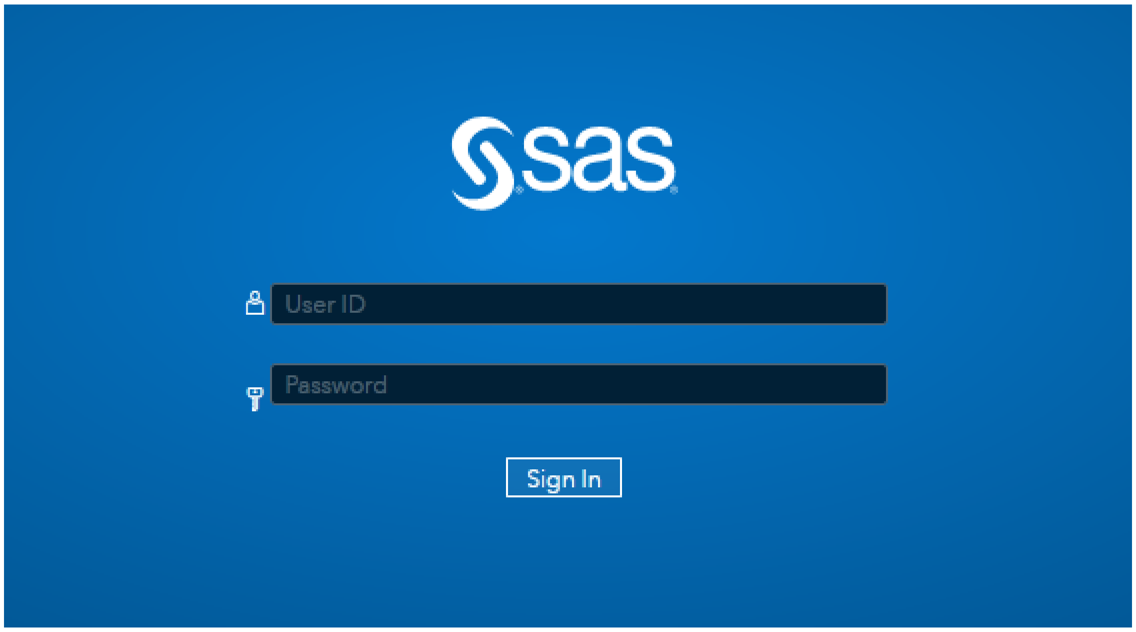


Figure 1

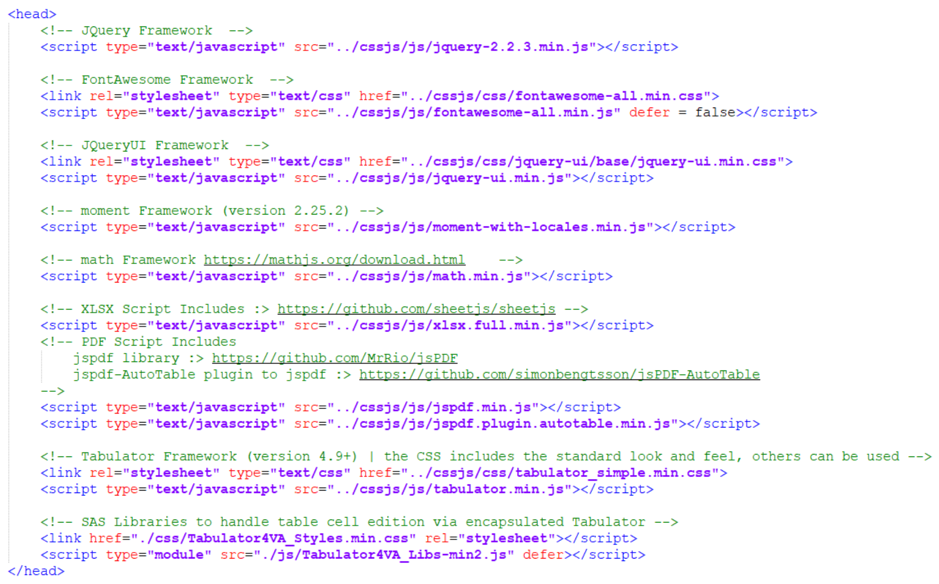
HTML Document Structure

Here is presented / described the structure of the **html that will be called by the DDC object**

```
<!DOCTYPE html>
<html>
    <head>
```

Figure 2

<head>

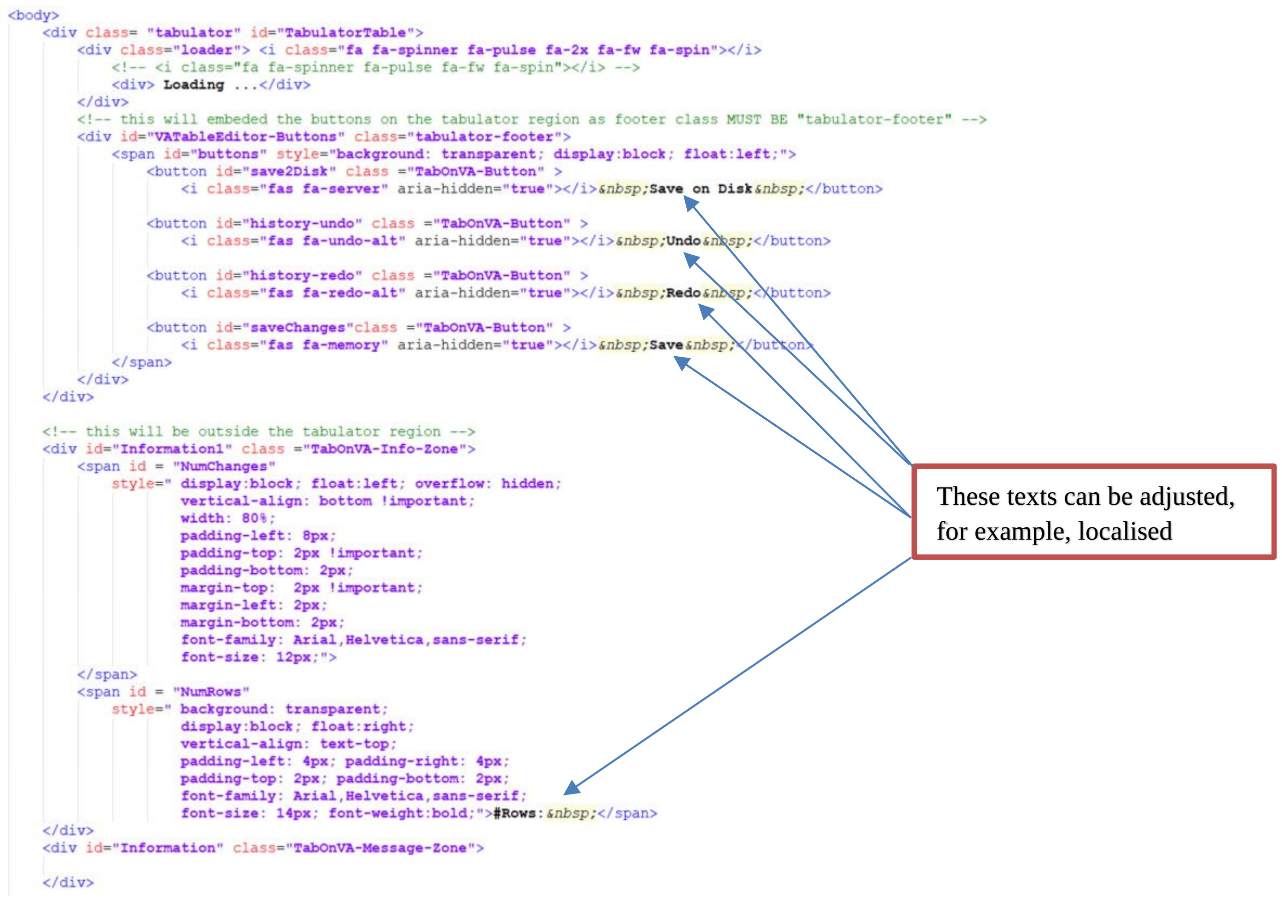


```
<head>
    <!-- JQuery Framework  -->
    <script type="text/javascript" src="../cssjs/js/jquery-2.2.3.min.js"></script>

    <!-- FontAwesome Framework  -->
    <link rel="stylesheet" type="text/css" href="../cssjs/css/fontawesome-all.min.css">
    <script type="text/javascript" src="../cssjs/js/fontawesome-all.min.js" defer = false></script>

    <!-- JQueryUI Framework  -->
    <link rel="stylesheet" type="text/css" href="../cssjs/css/jquery-ui/base/jquery-ui.min.css">
    <script type="text/javascript" src="../cssjs/js/jquery-ui.min.js"></script>

    <!-- moment Framework (version 2.25.2) -->
    <script type="text/javascript" src="../cssjs/js/moment-with-locales.min.js"></script>

    <!-- math Framework https://mathjs.org/download.html    -->
    <script type="text/javascript" src="../cssjs/js/math.min.js"></script>

    <!-- XLSX Script Includes :> https://github.com/sheetjs/sheetjs -->
    <script type="text/javascript" src="../cssjs/js/xlsx.full.min.js"></script>
    <!-- PDF Script Includes
        jspdf library :> https://github.com/MrRio/jsPDF
        jspdf-AutoTable plugin to jspdf :> https://github.com/simonbengtsson/jsPDF-AutoTable
    -->
    <script type="text/javascript" src="../cssjs/js/jspdf.min.js"></script>
    <script type="text/javascript" src="../cssjs/js/jspdf.plugin.autotable.min.js"></script>

    <!-- Tabulator Framework (version 4.9+) | the CSS includes the standard look and feel, others can be used -->
    <link rel="stylesheet" type="text/css" href="../cssjs/css/tabulator_simple.min.css">
    <script type="text/javascript" src="../cssjs/js/tabulator.min.js"></script>

    <!-- SAS Libraries to handle table cell edition via encapsulated Tabulator -->
    <link href="./css/Tabulator4VA_Styles.min.css" rel="stylesheet"></script>
    <script type="module" src="./js/Tabulator4VA_Libs-min2.js" defer></script>
</head>
```

Figure 3

<body>

<div> ‘s

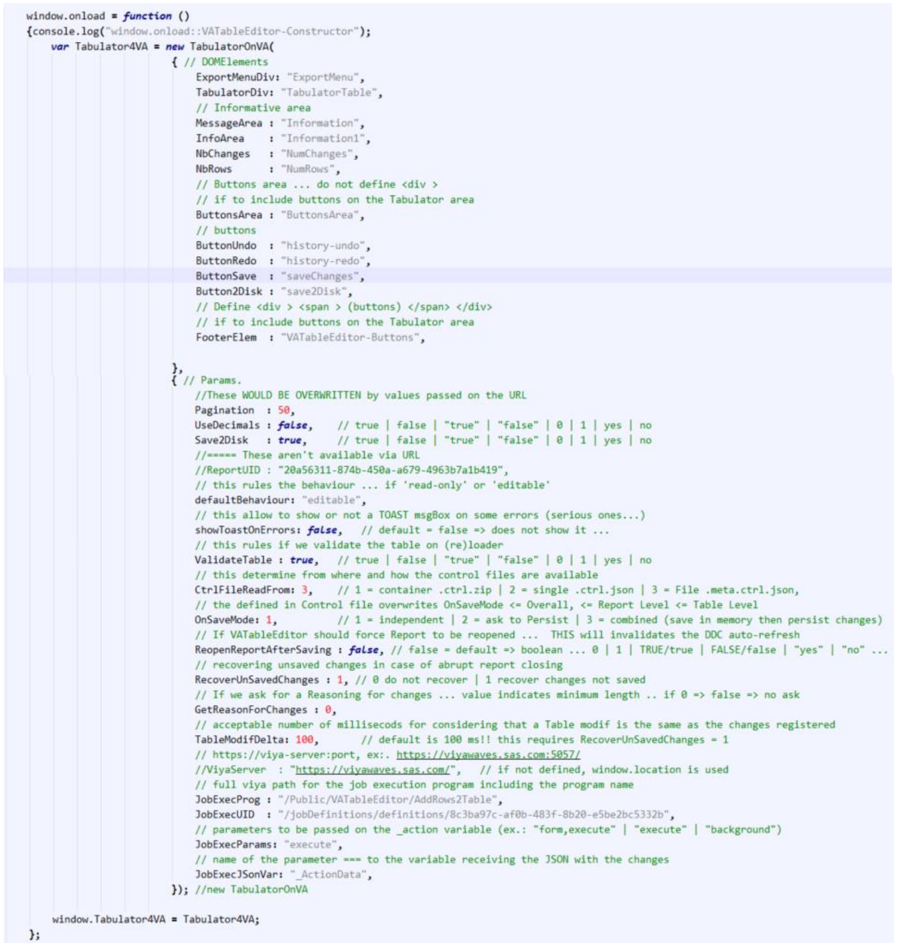


```
<body>
    <div class= "tabulator" id="TabulatorTable">
        <div class="loader"> <i class="fa fa-spinner fa-pulse fa-2x fa-fw fa-spin"></i>
            <!-- <i class="fa fa-spinner fa-pulse fa-fw fa-spin"></i> -->
            <div> Loading ...</div>
        </div>
        <!-- this will embeded the buttons on the tabulator region as footer class MUST BE "tabulator-footer" -->
        <div id="VATableEditor-Buttons" class="tabulator-footer">
            <span id="buttons" style="background: transparent; display:block; float:left;">
                <button id="save2Disk" class ="TabOnVA-Button" >
                    <i class="fas fa-server" aria-hidden="true"></i>&nbsp;Save on Disk&nbsp;</button>

                <button id="history-undo" class ="TabOnVA-Button" >
                    <i class="fas fa-undo-alt" aria-hidden="true"></i>&nbsp;Undo&nbsp;</button>

                <button id="history-redo" class ="TabOnVA-Button" >
                    <i class="fas fa-redo-alt" aria-hidden="true"></i>&nbsp;Redo&nbsp;</button>

                <button id="saveChanges"class ="TabOnVA-Button" >
                    <i class="fas fa-memory" aria-hidden="true"></i>&nbsp;Save&nbsp;</button>
            </span>
        </div>
    </div>

    <!-- this will be outside the tabulator region -->
    <div id="Information1" class ="TabOnVA-Info-Zone">
        <span id = "NumChanges"
            style=" display:block; float:left; overflow: hidden;
                    vertical-align: bottom !important;
                    width: 80%;
                    padding-left: 8px;
                    padding-top: 2px !important;
                    padding-bottom: 2px;
                    margin-top:  2px !important;
                    margin-left: 2px;
                    margin-bottom: 2px;
                    font-family: Arial,Helvetica,sans-serif;
                    font-size: 12px;">
        </span>
        <span id = "NumRows"
            style=" background: transparent;
                    display:block; float:right;
                    vertical-align: text-top;
                    padding-left: 4px; padding-right: 4px;
                    padding-top: 2px; padding-bottom: 2px;
                    font-family: Arial,Helvetica,sans-serif;
                    font-size: 14px; font-weight:bold;">#Rows:&nbsp;</span>
    </div>
    <div id="Information" class="TabOnVA-Message-Zone">

    </div>
```

These texts can be adjusted, for example, localised

Figure 4

<script>



```
window.onload = function ()
{console.log("window.onload::VATableEditor-Constructor");
    var Tabulator4VA = new TabulatorOnVA(
                        { // DOMElements
                          ExportMenuDiv: "ExportMenu",
                          TabulatorDiv: "TabulatorTable",
                          // Informative area
                          MessageArea : "Information",
                          InfoArea    : "Information1",
                          NbChanges   : "NumChanges",
                          NbRows      : "NumRows",
                          // Buttons area ... do not define <div >
                          // if to include buttons on the Tabulator area
                          ButtonsArea : "ButtonsArea",
                          // buttons
                          ButtonUndo  : "history-undo",
                          ButtonRedo  : "history-redo",
                          ButtonSave  : "saveChanges",
                          Button2Disk : "save2Disk",
                          // Define <div > <span > (buttons) </span> </div>
                          // if to include buttons on the Tabulator area
                          FooterElem  : "VATableEditor-Buttons",

                        },
                        { // Params.
                          //These WOULD BE OVERWRITTEN by values passed on the URL
                          Pagination  : 50,
                          UseDecimals : false,    // true | false | "true" | "false" | 0 | 1 | yes | no
                          Save2Disk   : true,     // true | false | "true" | "false" | 0 | 1 | yes | no
                          //===== These aren't available via URL
                          //ReportUID : "20a56311-874b-450a-a679-4963b7a1b419",
                          // this rules the behaviour ... if 'read-only' or 'editable'
                          defaultBehaviour: "editable",
                          // this allow to show or not a TOAST msgBox on some errors (serious ones...)
                          showToastOnErrors: false,   // default = false => does not show it ...
                          // this rules if we validate the table on (re)loader
                          ValidateTable : true,   // true | false | "true" | "false" | 0 | 1 | yes | no
                          // this determine from where and how the control files are available
                          CtrlFileReadFrom: 3,    // 1 = container .ctrl.zip | 2 = single .ctrl.json | 3 = File .meta.ctrl.json,
                          // the defined in Control file overwrites OnSaveMode <= Overall, <= Report Level <= Table Level
                          OnSaveMode: 1,          // 1 = independent | 2 = ask to Persist | 3 = combined (save in memory then persist changes)
                          // If VATableEditor should force Report to be reopened ...  THIS will invalidates the DDC auto-refresh
                          ReopenReportAfterSaving : false, // false = default => boolean ... 0 | 1 | TRUE/true | FALSE/false | "yes" | "no" ...
                          // recovering unsaved changes in case of abrupt report closing
                          RecoverUnSavedChanges : 1, // 0 do not recover | 1 recover changes not saved
                          // If we ask for a Reasoning for changes ... value indicates minimum length .. if 0 => false => no ask
                          GetReasonForChanges : 0,
                          // acceptable number of millisecods for considering that a Table modif is the same as the changes registered
                          TableModifDelta: 100,       // default is 100 ms!! this requires RecoverUnSavedChanges = 1
                          // https://viya-server:port, ex:. https://viyawaves.sas.com:5057/
                          //ViyaServer  : "https://viyawaves.sas.com/",   // if not defined, window.location is used
                          // full viya path for the job execution program including the program name
                          JobExecProg : "/Public/VATableEditor/AddRows2Table",
                          JobExecUID  : "/jobDefinitions/definitions/8c3ba97c-af0b-483f-8b20-e5be2bc5332b",
                          // parameters to be passed on the _action variable (ex.: "form,execute" | "execute" | "background")
                          JobExecParams: "execute",
                          // name of the parameter === to the variable receiving the JSON with the changes
                          JobExecJSonVar: "_ActionData",
                        }); //new TabulatorOnVA

    window.Tabulator4VA = Tabulator4VA;
};
```

Figure 5

```
</html>
```

Figure 6

Initializing the Object - Tabulator4VA

The object must be initialized on the window load by defining a response function to the event **window.onload**

Class Constructor Parameters

Html Document objects that will be used by the component:

{
TabulatorDiv : "<div id> where the Tabulator Table will be created",
ExportMenuDiv: "<div id> where the Export dropdown menu will be created",
MessageArea : "<div id> where the actions information messages will be displayed",
InfoArea : "<div id> where the additional information messages will be displayed",
NbChanges **:** "<div id> where the Number of Changes information will be presented",
NbRows **:** "<div id> where the Number of Rows will be presented",
ButtonsArea **:** "<span id> of the button for the history-undo action",

ButtonUndo : "<button id> of the button for the history-undo action",
ButtonRedo : "<button id> of the button for the history-redo action",
ButtonSave : "<button id> of the button for the Save Changes action",
Button2Disk : "<button id> of the button for the Save on Disk (persisting) the changes",
FooterElem : "<div id> for the footer, meaning, where the above will be presented",
},
See figure 4 and 5.

Execution Parameters used by the component:

{
//== These MIGHT BE OVERWRITTEN by values passed on the URL
Pagination : 50, // initial pagination size. That will derive in 1x, 2x, 3x and 4x
//== Should be equal to Column Label / Physical Name. Only used in case of unable to read/ find the primaryKey / Row Unique ID from the Report definition. The name **Unique_ID** is taken by default to match what SAS VA does.
RowUIDCol : "Unique_ID",
//== if we'll use decimals for rounding the value when presenting it
UseDecimals : false, // true | false | "true" | "false" | 0 | 1 | yes | no
//== if we let the user persisting changes by saving those to disk
Save2Disk : true, // true | false | "true" | "false" | 0 | 1 | yes | no
//== These aren't available via URL, but are **mandatory**
// this rules the behaviour ... if 'read-only' or 'editable'
defaultBehaviour: "editable",
// this allow to show or not a TOAST msgBox on some errors (serious ones)
showToastOnErrors: false, // default = false => does not show it ...
// if we validate the table on (re)load the following when defined in Control file overwrites <= Overall, <= Report Level <= Table Level
ValidateTable : true, // true | false | "true" | "false" | 0 | 1 | yes | no
CtrlFileReadFrom: 3, // 1 = container .ctrl.zip | 2 = single .ctrl.json | 3 = File .meta.ctrl.json,
OnSaveMode: 1, // 1 = independent | 2 = ask to Persist | 3 = combined (in memory + persist)
TableLayout : "fitColumns", //"fitData" | "fitDataFill" | "fitDataStretch" | "fitDataTable" | "**fitColumns**" <= default (see dataSources - array with the objects defining the data sources:)
RecoverUnSavedChanges : 1, // 0 do not recover | 1 recover changes not saved
// If we ask for a Reasoning for changes ... value indicates minimum length .. if 0 => false => no ask
GetReasonForChanges : 0,
// acceptable number of milliseconds for considering that a Table modif is the same as the changes registered
TableModifDelta: 100, // default is 100 ms!! this requires RecoverUnSavedChanges = 1
// Max number of pings to determine the state os a called Service, e.g. a JobExecution
ExtSrvStatusMaxPings : 100, // default is 30
ViyaServer : " https://your_server:port", // if omitted, is taken the server where VA is running
// Full viya path for the job execution program including the program name..
JobExecProg : "full path of your Job Execution .sas code",
JobExecUID : "/jobDefinitions/definitions/<your job unique id>",
// Parameters to be passed on the ***_action*** variable (ex.:"form,execute" | "execute" | "background")
JobExecParams : "execute",
// Name of the parameter **equal** to the variable receiving the JSON with the changes
JobExecJSonVar : "_ActionData",

}
See Figure 5

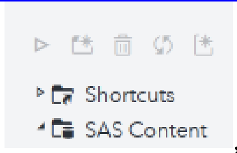
Setting the JobExecution(*) Parameters

JobExecProg

Look at the Job Execution Installation instructions to get the information. The location should be as:
JobExecProg : *"/Public/VATableEditor/AddRows2Table",*

JobExecUID

Normally Viya when importing / installing from the provided JSON, will preserve the Unique IDs. Thus, this step is more educational or to verify if is all good.

1. In a new Browser TAB, put **https://<your_server:port>/SASJobExecution/**
2. Click on the **'SAS Content'** to open it .. 

, scroll down to subfolder **'Public'** click on, scroll down till **'VATableEditor**, click on, look for **'AddRows2Table'**
3. With the mouse/pointer, Right-Click over **'AddRows2Table'**, on the presented menu, choose **'Properties'**
4. On **'General'** check if the **'Name'** matches with what you defined prior for the **JobExecProg** parameter
5. Click on **'Details'**, and copy the 'Job Details' (the '/jobDefinitions/…)
Paste that on the **JobExecUID : **"/jobDefinitions/definitions/<your job unique id>" and save the **.html** file.
6. Click button 'Cancel' on the bottom-right of the window.

Destroying the Object - Tabulator4VA

It's recommended to destroy the object on the window unload by defining a response function to the event **window.onunload** . That should be done by calling the **Exit** function.

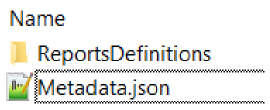
```
window.onunload = function (){
        <your Tabulator4VA object>.Exit();
};
```
See Figure 5

Setting Up the Control File

The control file used (see below as three different options are possible) **MUST BE** installed on the server on the same location as the pointed HTML file.

Control File / Definitions

From version 4.0, are three ways where the control file(s) are searched for loading:



1. *From a ZIP: VATableEditor.ctrl.zip*

   which must contain:

   a. ***Metadata.json***

```
{
    "information": {
            "VATableEditor-CtrlDefinitions" : "v2",
            "creationDate"              : "14Nov2020 12:50",
            "createdBy"                 : "user 6 letters uid",
            "lastModifDate"             : "14Nov2020 12:50",
            "lastModifBy"               : "user 6 letters uid",
            "TrackingTblLib"            : "caslib",     <= CAS Lib where table will be
            "TrackingTable"             : "VATableEditor_Changes",  <= Table Name
            "TrackChanges"              : true,         <= true | false | 0 | 1 | yes | no
            "defaultBehaviour"          : "read-only",  <= "read-only" | "editable"
            "ValidateTable"             : true,         <= true | false | 0 | 1 | yes | no
            "OnSaveMode"                : 1             <= 1 | 2 | 3 (see above ..)
            "RecoverUnSavedChanges"     : 1             <= 1 | 2  (see above ..)
    },
    "validUsers": {
            "admins" : {
                    "groups": [
                                "group1",
                                "...",
                                "group_n"
                            ],
                    "users": [
                                "user1",
                                "...",
                                "user_n"
                            ]
            },
            "groups": [
                    "group1",
                    "...",
                    "group_n"
            ],
            "users": [
                    "user1",
                    "...",
                    "user_n"
            ]
    },
    "reports": {
            "<Report_1 UID>": "<control full path+name within the ZIP>.json",
            (…)
            "<Report_n UID>": "<control full path+name within the ZIP>.json"
    }
}
```

b. *One or more <name>.json* => corresponding to "<control full path+name within the ZIP>.json". On the example, those are in the ZIP folder **'ReportsDefinitions'**

2. *From a unique JSON file: VATableEditor.ctrl.json*

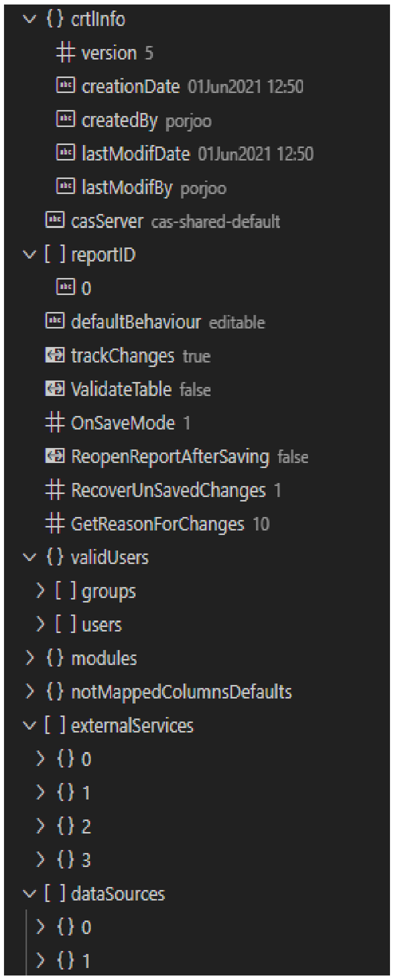
The control file is a JSON document composed as illustrated by the picture. An example is provided on the installation ZIP.

- **information**

  Identical to the object 'information' described above for option 1

- **reports array**

  Contain an object for each report, composed by:



```
v {} crtlInfo
    # version 5
    creationDate 01Jun2021 12:50
    createdBy porjoo
    lastModifDate 01Jun2021 12:50
    lastModifBy porjoo
  casServer cas-shared-default
v [ ] reportID
    0
  defaultBehaviour editable
  trackChanges true
  ValidateTable false
  # OnSaveMode 1
  ReopenReportAfterSaving false
  # RecoverUnSavedChanges 1
  # GetReasonForChanges 10
v {} validUsers
  > [ ] groups
  > [ ] users
> {} modules
> {} notMappedColumnsDefaults
v [ ] externalServices
  > {} 0
  > {} 1
  > {} 2
  > {} 3
v [ ] dataSources
  > {} 0
  > {} 1
```

- **ctrlInfo**

Object containing refence information about the definitions

- **casServer**

- **reportID**

an array with the reports UID that the file serves.

- **defaultBehaviour**

what would be the behaviour on this report for tables not having definitions: "editable" or "read-only"

- **trackChanges**

if on this report all changes saved will be tracked / registered = true, false otherwise

- **OnSaveMode**

Save mode for this report: **1** = independent, **2** = user is asked if want also to save in disk (persist the changes), **3** = combined mode (saving in memory also persist the changes in disk)

- **ValidateTable**

if on this report the tables will be validated on (re)Load

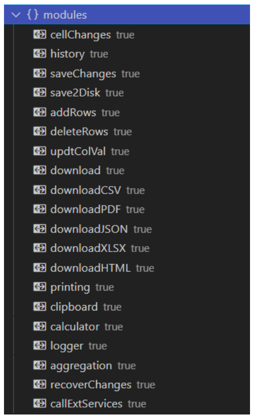
- **RecoverUnSavedChanges:**

If the App try or not to recover some unsaved changes. This depends on how the browser is configured, meaning that the lifetime of the unsaved changes is as long as the lifetime of the browser cookies.

- **validUsers**

Object containing an array with all the valid groups of users (where the user logged in on SAS VA belongs to) and another array with all the userID.
The "admins", when defined, VATableEditor will use that and if a user belongs to one of the **groups** and/or is referred as one of the **users**, then that is taken into consideration and some functionalities are

made available, which aren't for other type of users, for example: accessing the App Log, accessing the controlFile content, …
<u>If the logged user doesn't belong to one of the groups nor is on the list, the VA Table Editor changes to Read-Only mode!</u>



```
{} modules
  cellChanges true
  history true
  saveChanges true
  save2Disk true
  addRows true
  deleteRows true
  updtColVal true
  download true
  downloadCSV true
  downloadPDF true
  downloadJSON true
  downloadXLSX true
  downloadHTML true
  printing true
  clipboard true
  calculator true
  logger true
  aggregation true
  recoverChanges true
  callExtServices true
```

- **modules**

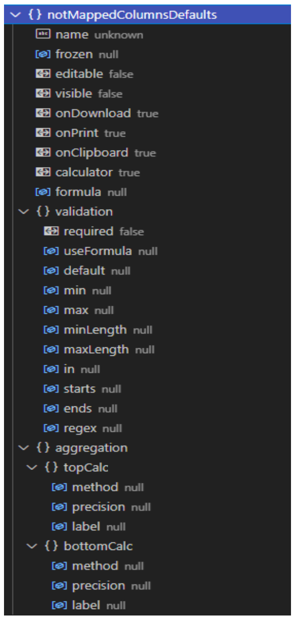
Object storing the modules valid / active, true = active | false not active.

If 'download' item is set to 'false' none of the download options will be activated!

If 'callExtServices' item is set to 'false' none of the defined ***external Services*** will appear on the **ContextMenu nor** will be executed upon saving!

- **notMappedColumnsDefaults**

object storing the parameters used to the columns not having a correspondent on the physical table. It contains another object **'validation'** which has the content validation parameters for the column.



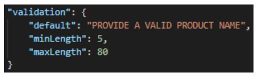
```
{} notMappedColumnsDefaults
  name unknown
  frozen null
  editable false
  visible false
  onDownload true
  onPrint true
  onClipboard true
  calculator true
  formula null
  {} validation
    required false
    useFormula null
    default null
    min null
    max null
    minLength null
    maxLength null
    in null
    starts null
    ends null
    regex null
  {} aggregation
    {} topCalc
      method null
      precision null
      label null
    {} bottomCalc
      method null
      precision null
      label null
```

- ***name :*** "unknown", or any other name.
- ***frozen :*** true | false > if the cell will be freeze, either on the left or right side of the table.
- ***editable :*** true | false > if the cell can be editable or not
- ***visible :*** true | false > if the cell will be presented or not
- ***onDownload:*** true | false > if the cell will be or not on the export file
- ***onPrint :*** true | false > if the cell will appear or not on the printing
- ***onClipboard :*** true | false > if the will appear or not on the copy to clipboard
- ***calculator :*** true | false > if the calculator (using formulas) can be used or not for getting cell value. Only applies for numeric cells.

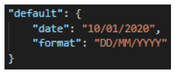
- **presentAs** : object containing the definitions on how to present **numeric column cells.**
  - ***type***: one of: **"stars", "checkbox", "progressBar", "trafficLights"**

  additional parameters for:
    - ***stars :***
      - ***max :*** *maximum number of stars => max value*
    - ***checkbox:***
      - ***tickElement:*** *see below…*

        - ***crossElement :*** *font awesome icon class, e.g.: "<i class='fa fa-check'></i>"*
    - ***progressBar and trafficLights:***
        - ***min:***
        - ***max:***
        - ***color :*** *[color$_1$, .., color$_n$] this can be , HTML5 "colour name", "RGB(r,g,b)" or "hsl(r%, g%, b%)". The 100% will be divided in as many chunks as number of colours, being the bar colour the correspondent one.*
    - ***editor:*** *if **null** or not defined, changing changes cell value by directly interaction with the representation. If defined **must** be "numeric"**. <u>In case the cell has defined "editorSelect" that is what will be used !</u>***

- **validation**
    - ***required*** : true | false
    - ***useFormula*** : true | false   > if the column formula will be used for validation
    - ***default*** : used for when detected an error on content provided.
        - for other data type string and numeric, provide the default value.
            - Example: 

```
"validation": {
    "default": "PROVIDE A VALID PRODUCT NAME",
    "minLength": 5,
    "maxLength": 80
}
```

        - for date/datetime, an object **{ "date" : <string with the date>, "format" : < 'momentjs' format string> }**.
            - Example : 

```
"default": {
    "date": "10/01/2020",
    "format": "DD/MM/YYYY"
}
```

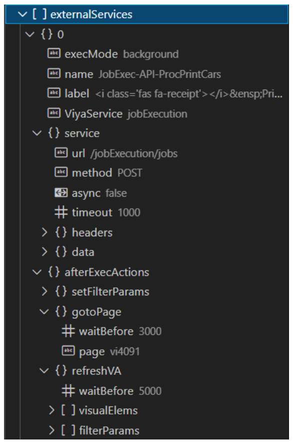
        - If no default provided, the App will default
            - Numeric: **0** (zero)
            - String: **"??????"**,
            - Date: **current date** in the format **DD/MM/YYYY**
            - Time: **current time** in format HH:mm:ss
    - ***min*** : for numeric cells. Minimum value accepted on the cell. Can receive a formula
    - ***max*** : for numeric cells . Maximum value accepted on the cell. Can receive a formula
    - ***minLength*** : for string cells. Minimum length the string should have
    - ***maxLength*** : for string cells. Maximum length the string should have
    - ***in*** : for string cells . A '|' separated string containing the valid values. Examples: "Y|N" or "Red|Blue|Green" …
    - ***starts*** : used for string. The initial characters the string must contain
    - ***ends*** : used for string. The final characters the string must contain
    - ***regex*** : used for numeric or string cells. Regular expression that validates content.

- **aggregation**

  the bellow applies for both **topCalc** and **bottomCalc**

  - ***method*** : aggregation function, please see Aggregation functions section
  - ***precision*** : decimal to use on the returned calculated value
  - ***label*** : what will be displayed on aside of the calculated value. If null or not present, the label will be the method (function name). is possible to use HTML formatting like use of font awesome icons, see report control file example provided.

externalServices



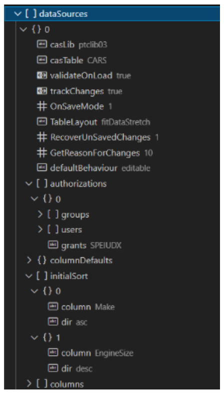
```
[ ] externalServices
  { } 0
    execMode  background
    name  JobExec-API-ProcPrintCars
    label  <i class='fas fa-receipt'></i>&ensp;Pri...
    ViyaService  jobExecution
    { } service
      url  /jobExecution/jobs
      method  POST
      async  false
      timeout  1000
      { } headers
      { } data
    { } afterExecActions
      { } setFilterParams
      { } gotoPage
        waitBefore  3000
        page  vi4091
      { } refreshVA
        waitBefore  5000
        [ ] visualElems
        [ ] filterParams
```

- **execMode: <u>ONLY applies on '*ctxMenuExtServices*'</u>**
  - "background" Service is launched silently ||
  - "foreground" is launched in a new Window.
- **name:** name of the service, this is used on the logging information
- **label:** what will be presented on the context menu. It can include *font awesome* icons
- **ViyaService: Use only if** a Viya Service (like SASJobExecution, jobExecution, …). See examples.
- **service:** contains the definitions for the **ajax** XHR request
  - **url:** URL of the service to be called.
  - **method:** "GET", "POST" (default), … other HTTP method
  - ***async:*** *true (default) || false*
  - ***timeout:*** *max time, in milliseconds, to wait for completion*
  - **(…) – see JQuery ajax XHR request documentation**
  - **headers**: object containing the request header definitions
  - **data:** object containing the data to be sent to server. It can contain other objects and/or arrays. Property value can make use of formulas, see examples.
- **afterExecActions:** contains the definitions of what will be executing after the service being completed. ***The sequence matters!!*** <u>The actions will be executed on the sequence that are defined!!</u>
  - **setFilterParams:** {<param name> : <value to set>, (…)} Where <value to set> can be *calculated*, i.e.: a formula (using the available functions) can be used to get the parameter value.
  - **refreshVA:** contains the name of the visual elements and/or the filterParams that should be refreshed after executing the service.
    - **waitBefore:** time in milliseconds before kicking the refresh
    - **visualElems:** array with a list of report visual elements (can be obtained from report BIRD)
    - **filterParams:** array with a list of parameters to be refreshed (are the names defined on the report)
  - **gotoPage:** definition for what page should "jump" to
    - **waitBefore:** time in milliseconds before jumping
    - **page:** page name (NOT the label presented on the report) which can be found on the BIRD. Hidden pages are showed as "pop-up"

    The actions below, defined as mentioned above, can be part and will be execute on its sequence

  - **setFilterParams :** see above
  - **refreshVA :** see above

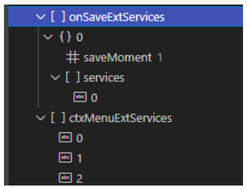
- **dataSources** - array with the objects defining the data sources:



```
[ ] dataSources
  {} 0
    casLib ptclib03
    casTable CARS
    validateOnLoad true
    trackChanges true
    OnSaveMode 1
    TableLayout fitDataStretch
    RecoverUnSavedChanges 1
    GetReasonForChanges 10
    defaultBehaviour editable
    [ ] authorizations
      {} 0
        [ ] groups
        [ ] users
        grants SPEIUDX
    {} columnDefaults
    [ ] initialSort
      {} 0
        column Make
        dir asc
      {} 1
        column EngineSize
        dir desc
    [ ] columns
```

- **casLib :** name of the cas library
- **casTable:** name of the cas table
- **validateOnLoad:** true if table should be validated on (re)Load, false otherwise
- **trackChanges:** true if saved changes will be tracked, false otherwise
- **OnSaveMode:** Saving mode for the table. <u>This overwrites previous definitions</u>
- **TableLayout :** Layout for the table. <u>This overwrites previous definitions</u>.
  - "fitData": will resize the tables columns to fit the data held in each column
  - "fitDataFill": as the fitData mode, but ensures that rows are always at least the full width of the table, may implying adding a void column at the end
  - "fitDataStretch": the table will resize the columns to fit their data and stretch the final column to fill up the remaining width of the table.
  - "fitDataTable" : will automatically resize container and columns to fit the data
  - "fitColumns" : the table will resize columns so that they fit perfectly inside the width of the container. Is the default used.
- **RecoverUnSavedChanges** : true / false (0 | 1) . if the App will recover unsaved changes in case of abrupt report closing and/or reopen. **Should be true** for a concurrent user deployment. <u>This overwrites previous definitions</u>
- **GetReasonForChanges** : 0 => no dialog box is presented to capture the Tweet like comment; any integer > 0 represents the minimum length for the tweet like comment that will be captured on the presented dialog box. <u>This overwrites previous definitions</u>
- **initialSort**: array containing one or more objects (as below). **The sequence matters!! <u>If not provided, table will be sorted ascendingly by the Row Unique ID column</u>**
  - **column:** <name of the column to sort>
  - **dir:** sort direction: "asc" => ascending | "desc » => descending
- **authorizations:** Array with the **grants** for the **groups** and/or **users**
  - **grants**: **S**ave **P**ersist **E**dit **I**nsert **U**pdate **D**elete e**X**port
    - THE SEQUENCE MATTERS Can be defined:
      - as a string with the letter on the respective position to allow, a '-' for disallow
      - as a string with binary representation (1= allowed | 0= disallowed)
      - as decimal equivalent to the binary representation

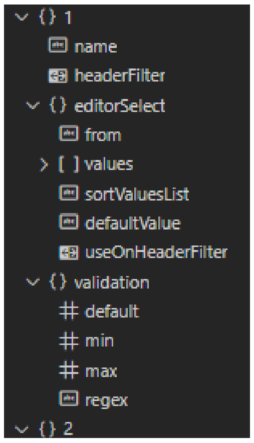
  - **onSaveExtServices** and **ctxMenuExtServices: <u>Depends on Module 'callExtServices'</u>**



```
[ ] onSaveExtServices
  {} 0
    saveMoment 1
    [ ] services
      0
[ ] ctxMenuExtServices
  0
  1
  2
```

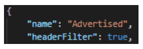
- **saveType: <u>ONLY applies on *'onSaveExtServices'*</u>**
  - 1 / "onSave" => when Saving in Disk
  - 2 / "onPersist" => when Saving on Disk.
- **services: <u>applies on *'ctxMenuExtServices'*</u>**
  - array with the names of the ***externalService*** to be executed. This MUST be defined on the **<u>externalServices</u>**
  - "foreground" is launched in a new Window.

- **columns:** Array containing table columns definitions
  - **columnDefaults:**



```
{} 1
    name
    headerFilter
    {} editorSelect
        from
        [ ] values
        sortValuesList
        defaultValue
        useOnHeaderFilter
    {} validation
        # default
        # min
        # max
        regex
{} 2
```

- this object has the same structure defined on notMappedColumnsDefaults. So, refer to it for the details.
- each column object has the same structure defined on **columnDefaults**, however, there's the possibility to make that on edition mode, the content being selected from a List of Values / Drop-Down List. For that the object **'editorSelect'** should be defined!
  - If you want to filter the columns directly from the table header:
- Should add / define the **'headerFilter'** column option as **'true'**

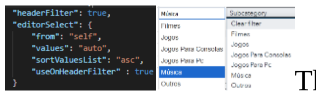


```
{
    "name": "Advertised",
    "headerFilter": true,
```

- If columns have also an **'editorSelect'** defined (see below) and if this option **'useOnHeaderFilter'** is set to **'true'**, **the same List of Values is taken as filter options!!**

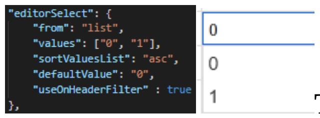
- **Option 1 – 'useOnHeaderFilter': true AND NO "editorSelect" defined !!**
  For numeric and strings columns, is possible to filter the column content using '=', '>', '>=', '<', '<=', '!=' ('<>') and 'like' (only for strings), 'in [V1, .., Vn], '!in [V1, .., Vn]', 'btw [min, .., max]', '!btw [min, .., max]', '<!> 0 | 1' . The default for numeric columns is '=' and for strings 'like'. For reset the filtering, clean the field/editor and press Enter.
  Example for numeric: >= 1.12 .. press Enter
  Example for string: > A .. press Enter

Options available for "editorSelect"

- **Option 2** – 

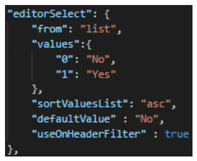
```
"headerFilter": true,
"editorSelect": {
    "from": "self",
    "values": "auto",
    "sortValuesList": "asc",
    "useOnHeaderFilter" : true
}
```

The list will be populated by the unique values exiting on the column
  - **"from"** – Indicates the type of source, this case **"self"**
    - **"values" – should be "auto"**
    - **"sortValuesList"** – can be **"asc"** for ascending or **"desc"** for descending. If not provided or **null**, will be the order found on the table
    - **"useOnHeaderFilter"** – if set to true **AND** the column has the option **'headerFilter' set to 'true'**, the same list of values is presented as header filter.

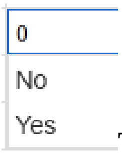
- **Option 3** – 

```
"editorSelect": {
    "from": "list",
    "values": ["0", "1"],
    "sortValuesList": "asc",
    "defaultValue": "0",
    "useOnHeaderFilter" : true
},
```

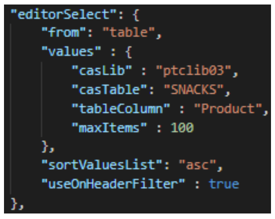
The list will be populated by given isolated values
  - **"from"** – Indicates the type of source, this case **"list"**
  - **"values" – should be an array** containing the list of elements
  - **"sortValuesList" –** can be **"asc"** for ascending or **"desc"** for descending. If not provided or **null**, will be the order found on the list.
  - **"defaultValue" –** Indicates the value by default. If not provided, will be the first on the list.
  - **"useOnHeaderFilter"** – if set to true **AND** the column has the option **'headerFilter' set to 'true'**, the same list of values is presented as header filter.

- **Option 4 –** 

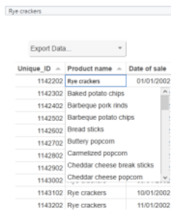
```
"editorSelect": {
    "from": "list",
    "values":{
        "0": "No",
        "1": "Yes"
    },
    "sortValuesList": "asc",
    "defaultValue" : "No",
    "useOnHeaderFilter" : true
},
```



The list will be populated by given pair of values, returned value, value presented.
    - **"from" –** Indicates the type of source, this case **"list"**
    - **"values" – should be an Object** containing the list elements.
        - The **key** (left side of ':') is the returned value (what will be stored on the cell)
        - The **value** (right side of ':') is the value presented on the list.
    - **"sortValuesList" –** can be **"asc"** for ascending or **"desc"** for descending. If not provided or **null**, will be the order found on the list.
    - **"defaultValue" –** Indicates the value by default. If not provided, will be the first on the list.
    - **"useOnHeaderFilter"** – if set to true **AND** the column has the option **'headerFilter' set to 'true'**, the same list of values is presented as header filter.

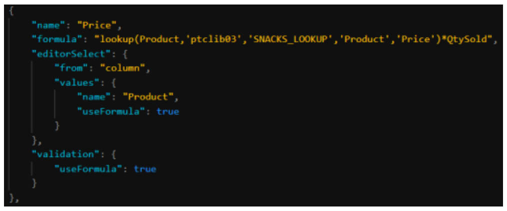
- **Option 5 –** 

```
"editorSelect": {
    "from": "table",
    "values" : {
        "casLib" : "ptclib03",
        "casTable": "SNACKS",
        "tableColumn" : "Product",
        "maxItems" : 100
    },
    "sortValuesList": "asc",
    "useOnHeaderFilter" : true
},
```



Unique_ID	Product name	Date of sale
1142202	Rye crackers	01/01/2002
1142302	Baked potato chips	
1142402	Barbeque pork rinds	
1142502	Barbeque potato chips	
1142602	Bread sticks	
1142702	Buttery popcorn	
1142802	Carmelized popcorn	
1142902	Cheddar cheese break sticks	
1143002	Cheddar cheese popcorn	
1143102	Rye crackers	10/01/2002
1143202	Rye crackers	11/01/2002

The list will be populated by the unique values exiting in a column from another table.
    - **"from"** – Indicates the type of source, this case **"self"**
    - **"values" – should be an Object** with the information to obtain the values
        - **"casLib" – CAS Library** where the table is stored
        - **"casTable" – CAS Table** from where the data will be read
        - **"tableColumn" – Column** on the table containing the values
        - **"maxItems"** – **Max** number of items on the list. If not provided, all **Column** distinct values will be presented.
    - **"sortValuesList"** – can be **"asc"** for ascending or **"desc"** for descending. If not provided or **null**, will be the order found on the table
    - **"useOnHeaderFilter"** – if set to true **AND** the column has the option **'headerFilter' set to 'true'**, the same list of values is presented as header filter.

- **Option 6 –** 

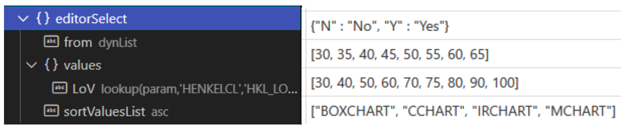
```
{
    "name": "Price",
    "formula": "lookup(Product,'ptclib03','SNACKS_LOOKUP','Product','Price')*QtySold",
    "editorSelect": {
        "from": "column",
        "values": {
            "name": "Product",
            "useFormula": true
        }
    },
    "validation": {
        "useFormula": true
    }
},
```

The cell will receive its value from another column, either from the same row, or from an external lookup table (example in the picture)

<u>Will be not possible to manually edit a cell receiving its value from another. The cell become **read-only!**</u>
    - **"from"** – Indicates the type of source, this case **"column"**
    - **"values" – should be an Object** with the information to obtain the values
        - **"name" – column name** supplying the value.
            - If multiple columns, like **"name": ["col1", "cols2", …]**

  - ▪ **"useFormula" – true | false** determines is the formula defined for the column will be used when applying the value coming from the source column.

- array with the definitions for the columns that will be presented. <u>Although a column being selected on VA Report > Roles, if not present here, the column will not be on the VA-TableEditor!</u>
- The definitions here will prevail as will be merged with the '**columnDefaults**'.
- If no columns on the array nothing will be presented.
- The object must contain at least the 'name'



editorSelect
from dynList
values
LoV lookup(param,'HENKELCL','HKL_LO...
sortValuesList asc

{"N" : "No", "Y" : "Yes"}
[30, 35, 40, 45, 50, 55, 60, 65]
[30, 40, 50, 60, 70, 75, 80, 90, 100]
["BOXCHART", "CCHART", "IRCHART", "MCHART"]

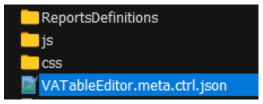
- **Option 7 – from a dynamic list**
  This option is supported on the use of the 'lookup' function in which the returned column should return values in a shape of a JSON object or an array of values.. like the options for 'List' (Option 3 and 4).

○

○ **Primary Key Definitions**

- If the column is primary key, MUST be present (see previous bullet) on the array and the object item '*primaryKey*' should be set to 'true'! If you want to hide the primary key column, set the object item 'visible' to 'false'! <u>This ONLY WORKS if this column matches with the Primary Key column defined on the VA report</u>.
- The Primary Key column will be always present on the 'Clipboard' actions – copy to | paste from!
- The Primary Key column will be present on the 'Printing' and/or on the 'Download' (aka Export) if the respective object items are set to 'true'!
- If any column needs special validations, the '**validation**' object must be set. Please refer to **columnDefaults** for the details regarding the object items.

*3. Based on a Metadata file: **VATableEditor.meta.ctrl.json** and individual control files read from the file system.*



ReportsDefinitions
js
css
VATableEditor.meta.ctrl.json

- ***VATableEditor.meta.ctrl.json*** structure is equal to the described on <u>option 1</u>
- The individual control files have the same structure defined on <u>option 2</u> for each element of the <u>reports array</u> . Its physical location should then be the one defined for the report(s) on the ***VATableEditor.meta.ctrl.json*** as per described for the <u>option 1</u>.

USING THE COMPONENT

Setting up a Data Driven Content in a SAS© VA Report

Creating a VA New Report

Access to your SAS Viya environment using your credentials, on the top-left corner clicking on "hamburger" icon you will get a menu like Figure 7 presented below, then you click on 'Explore and Visualize'.

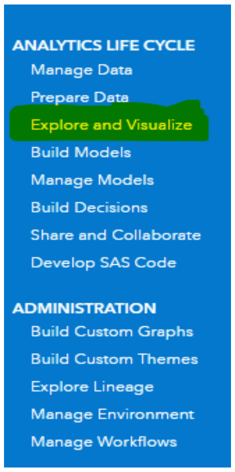


Figure 7

Create a **New Report**, by clicking on the respective button

Linking / Selecting the Table

The table(s) that will be used on the with Tabulator4VA and object to be edited, **must have a column with the row unique identifier**, preferably a numeric.

On the left-hand menu, click on Data and select your table from the drop-down list

Defining the Primary Key - MANDATORY

1. The Primary Key, although working perfectly if a string, it's recommended to be a numeric (long integer, int64, ...)

2. Once the table is linked, you should see its columns classified either as 'Category' or 'Measure'.
3. The column which contains the unique row identifier, should be a 'Category', if isn't, select it and change its classification to 'Category'.

4. Click with the mouse right button on the column, and on the presented menu, click on **'Create row unique identifier' – Check '<u>Setting Up the Control File</u>' regarding the <u>primary key options</u>.**

Selecting the Filters – Columns to filter the displayed rows

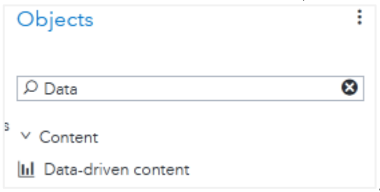
If at this point you already know what columns would be acting as filters, it's time to define the 'Parameters', otherwise you can get to this step later.

1. Click with the mouse right button on the column you want to use as filter, and on the presented menu, click on **'New Parameter'**
2. Give a name that you can identify and relate with the purpose (you may want to accept the proposed name)
3. Clean the content on 'Minimum Value', 'Maximum Value' and 'Current Value' and click **OK**

4. Repeat steps 1 .. 3 for all columns you are going to use as filters

5. Save the Report by clicking on the icon on the top right of the window.

Create / Insert the Data Driven Content on the report

1. Again, on using the left-hand menu, click on the icon and, either scroll down on the presented menu until you find **'Data-driven content'**, or on the filter field write 'Data' and you will get the one only option 

.
2. Select the option and drag it to the report area dropping on the zone you want

Selecting the Columns to be Presented

Check 'Setting Up the Control File'

1. On the Right-Hand menu, click on the icon and add all the variables/columns you want to present and would be subject to be edited.
2. By dragging the selected columns, you can put them in the sequence you want.
3. If you select a column which hasn't a correspondent table column, the cells of that column can't be edited!
5. You must select the column containing the row unique identifier! **This column is not available for edition! Pay attention to 'Setting Up the Control File' regarding the primary key options.**

Pointing / assign the HTML document

1. On the Right-Hand menu, click on the icon and on the 'Web Content URL' provide the full URL for your html defined above. That should be something like
   https://<your_server:port>/htmlcommons/DataDriven/Tabulator4VA/DDC_Tabulator4VA_(2).html
   a. Optionally you can add some options/parameters on the URL. See Figure 5
      i. **PageSize**=<number of rows per page>
      ii. **UseDecimals** = <Boolean> (true | false | "true" | "false" | yes | no | 0 | 1)
      iii. **Save2Disk** = <Boolean> (true | false | "true" | "false" | yes | no | 0 | 1)
      EXAMPLE
      ?PageSize=50&UseDecimals=no&Save2Disk=yes
2. Press 'Enter'
3. Save the Report by clicking on the icon on the top right of the window.

Including Columns acting as Table Filters

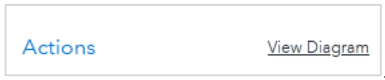
. The filters will then filter the rows sent to the DDC object – the VA-TableEditor – which gives then the possibility to present only the relevant rows.
If you define Filters, we strongly suggest using Dropdown Lists to be used as Table filter, as is an object that takes a small part of the visualization area. Thus:

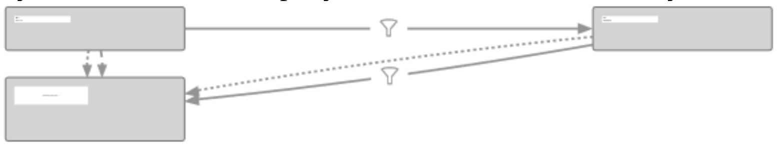
1. Using the left-hand menu, click on the icon and, either scroll down on the presented menu until you find **'Dropdown List'**, or on the filter field write 'Drop'
2. Select the option and drag it to the report area dropping on the zone you want
3. Repeat previous steps to add the other filter columns.
4. Save the Report by clicking on the icon on the top right of the window.

Assigning the Parameters to the columns filtering the rows

1. Select the **Dropdown list** you want to assign the parameter
2. On the Right-Hand menu, click on the icon Roles and on **Data Roles** at the bottom you see **Parameters**, click on the 'plus' **(+)** icon and add the corresponding parameter you have defined in **Selecting the Filters**. If at this stage you haven't yet defined those, it's now the time.

Establishing the dependency links between the Filters and the DDC Object

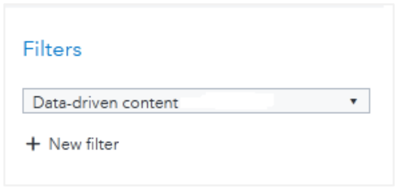
1. On the Right-Hand menu, click on the icon Actions
2. On the top of the opened feature are, you should have 

, so click on **View Diagram**.
3. On the opened window, you should have a few boxes like as the number of Dropdown Lists (or any other filter object chosen) you have created previously. In addition, you should also have an image like corresponding to the DDC object you defined.
   a. Starting on the 1st corresponding to the top filter, click on it (around the white rectangle) and arrow should appear, so drag it towards the next if exists, which should be the 2nd filter level. Repeat for al other you may have defined.
   b. From the last filter, do the same as above but this time the destination is the DDC Object .
   c. Once you finish, if for example you have two filter columns, you should have a picture like 
   d. Click on button Close at the bottom right.
4. Save the Report by clicking on the icon on the top right of the window.

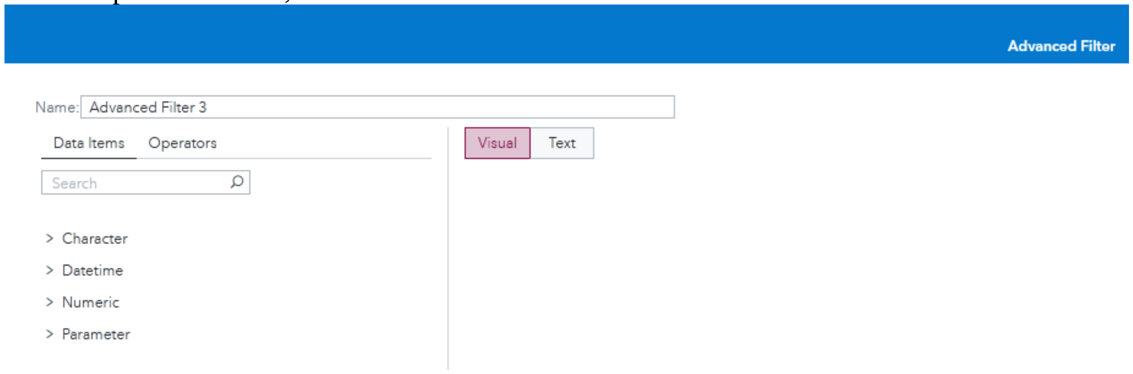
Exposing the Parameters to the Tabulator4VA DDC Object

Now we need to expose the Parameters linked to the filter columns do the DDC object. To do that:

1. Select the DDC Object

2. On the Right-Hand menu, click on the icon Filters, on the opened feature area you will see something like 

3. Click on **New Filter**. A list should appear at the left and at the top click on **Advanced Filter**
4. On the Opened window,



5. Give a name to the filter. We suggest being like the parameter you will be using.
6. On the **Data Items**, expand the **> Parameter** .. then you see the parameters you defined previously and that were also assigned to the filtering columns
7. On the Visual Text, select **Text** Visual Text
8. Clean any text that may appear below Visual Text. On the left side, where you have the parameters, double click on one (the sequence / order, isn't relevant), then you will have on the **Text** area something like **'ParameterA'p**.
9. Go to the **Text** area and position at the end of the text. Now type = and double-click again on the same parameter. You should end up with '**ParameterA'p = 'ParameterA'p**
10. Click on the **OK** at the bottom right of the window.
11. Repeat the process from the step 3, for all other parameters you might have.
12. Save the Report by clicking on the icon on the top right of the window.
13. **You done!**

USER MANUAL – EXPLORING THE COMPONENT

Functionality SUMMARY:

VATableEditor it allows to do changes at the cell level. For that multiple functionalities are available (some are module activation dependent):

- Header Filtering (in addition to the VA report filtering linked to the DDC object)
- Cell contents select from:
    - another cell
    - another cell via lookup another table for a value. it's possible to combine with a formula
    - another table,
    - column values itself ... this also available for Header Filtering
    - provided list of values (simple or label => value)
    - dynamic list of values supplied by a lookup
- Cell content can be calculated by a formula. This applies to all column types (numeric, date, time, datetime, strings)
- Add rows by duplicating an existing row and/or a defaulted row
- Table Pagination
- No handling limit on columns nor rows (the only limits are the ones imposed by VA)
- Applying a value to an entire column (respecting the filtering) with the possibility to Rollback
- Changes History Management (Undo / Redo). Applies to cell changes, row adding and/or deleting
- Table content validation (cell by cell). Intrinsic column type validation, VA variable definitions, configurable rules... Invalid cells log (with printing possibility) and violations exposed on cell tooltip and a suggested value is presented.
- Cell validation rules tested at input moment to avoid entering wrong values. Those rules can make use of formulas, for example testing the min | max based on a value of other cell.
- Ability to use formulas/expressions on editing cells (all types except date, time and datetime as for those pickers are provided).
- Table exporting data to multiple formats
    - CSV, PDF, JSON, EXCEL, HTML
    - Printer
    - Copy to Clipboard
- Ability to export to Clipboard > apply changes / add new rows > re-import table
- Application log export to CSV, JSON, HTML and Print option
- All Changes traceability registering in a tracing table. This at cell level detail including all details about the change made. Configurable from cell level to Report Level.
- Saving the changes on CAS Table and/or persist those changes on physical table. Three ways of controlling:
    - combined saving in memory and persisting
    - save in memory and asking if to persist
    - save in memory, persisting in a separated action
- Ability to call internal SAS Viya Services (like JobExecution) and/or 3rd party WebServices, in background or foreground. From the context menu and then passing information like the row UID, column data, row data, entire table data, modified rows, .. and upon the saving moments (in memory and in Disk)
- Column aggregation values. Two different aggregations are possible, on the top and/or other on the bottom. Function includes the most common statistical ones.
- Recovery of unsaved changes in case of closing the report. This is configurable on the HTML, control file, master and/or report control file. The recovery depends on how the browser is configured regarding the persistence of cookies.

- Column independent Header filtering (not dependent from the cell ***editorSelect*** config), in numeric and string/alphanumeric columns. Is possible to filter by '=', '>', '>=', '<', '<=', '!=' and 'like' (only for strings), ‘in’, ‘!in’, ‘btw’, ‘!btw’ (aka ‘out’), ‘<!> 1 | 0’ (1= for getting only the rows with invalid cells; 0 = otherwise)
- Ability to freeze columns, either on the left and/or right of the table. 4) Informative messages (at the bottom, underneath the buttons) coloring.
- External Services calls / executions are logged into the Table Tracker. Configurable!
- Saving the App log on the file system on **./Logs** folder upon closing report. The ./Logs folder needs to allow writing / creating files.
- If a cell content has changed based on the formula, that is presented as cell tooltip and if user clicks back on the cell for another edition, what's presented is the used formula and not its result.
- Multi-User : User Concurrence on Save
    - VATableEditor will check if table has been modified and if those modifications were made by another VATableEditor User or not. In case that was another VATableEditor User, those changes are assessed to find if there's any impact on the changes about to be saved.
- Security:
    - Align/sync with Viya Group/User Table Authorizations
    - Implementing Admin Group(s) and/or User(s)
    - Implement own authorization schema per table. This is defined at user group and/or user.
    - The functionalities are then aligned to the Authorization profile.

Using the Component

Editing individual cells

Each editable cell (all cells belonging to non-editable columns aren't editable), is subject to the respective data type:

- **Time, Date Datetime**:
    - Manually and/or through browser Time/Date Picker, which pops up clicking on one of the date components.
    - The date is validated, to avoid wrong dates, respecting the date informat defined on VA for the column.

- **Strings**:
    - The length defined on the column format / informat on VA, is tested.

- **Numeric**:
    - Integers:
        - The max length / value defined on the column format / informat on VA, is tested.
    - Floats:
        - The max length (including the decimals) defined on the column format / informat on VA, is tested.
        - If parameter '**UseDecimals**' is set to **true**, the displayed value respects the decimals defined on the column format / informat on VA, although all decimals are stored on the table.

Using formulas:

- it's possible use any type of math formulas to set the column value, please refer to https://mathjs.org/docs/reference/functions.html to see what's possible.
- When defining a formula, use the column visible name (column label), if it includes spaces and/or special characters, the column name (label) **MUST be enclosed** in **"** or **'**
- It's possible to use the same column with the formula and columns that aren't editable.
- A video showing an example it's provided on the ZIP file.

Additional functions: see Appendix 1

If the History module is active, Undo / Redo functionality are available. Thus, all cell changes are possible to revert and reapply in LIFO – Last In First Out.

Deleting Rows

For deleting a row, you click with the mouse right button, on the pop-up menu choose 'Delete Row'.
A deleted row can be recovered (LIFO) if the History module is active.

Applying a value to an entire column

For applying a value to all cells of a column, you click with the mouse right button, on the pop-up menu choose 'Apply to All.

After doing a mass change on a column, is still possible to change cells individually and are those the values that will prevail when saving the changes.

- The value is applied to all cells of the selected column respecting the selected filters. If there's column Header Filter applied, those are also taken into consideration. In case of a column being a VA report filter and also a header filter, the VA report filter content – having content – has priority over the header filter content. Thus, as the filters applied to the table only return a subset of rows, only those will get the new value.
- It's possible to revert by applying the Rollback option.
  - click with the mouse right button, on the pop-up menu choose 'Rollback'.
  - The individual cell changes made after the mass update aren't affected by the Rollback action.

Adding / Duplicating Rows

For Adding / Duplicate a row, click with the mouse right button, on the pop-up menu choose 'Duplicate Row'.

- A new row Unique ID is found and assigned to the new row, which will have all other cells identical to the source/copied row.
- The new row Unique ID it's valid for the current session and may change when inserting the row on the table, so, don't get panic if you do not find the generated row unique id on the table or mapping with the rest of the cell values.
- The new row is taken as any other row, therefore is possible to individually change cell values, participate on an eventual column mass value change.

If the History module is active, Undo / Redo functionality are available. Thus, all cell changes are possible to revert and reapply in LIFO as well as undoing / redoing the addition of the row.
If the new row cells are changed, those changes will be respecting the LIFO.

History Management – Undo / Redo

The History Management stores all actions made over the cells and rows. Access to the history is made by the buttons ⟲ Undo ⟳ Redo respecting Last In First Out.

Saving changes to the in-memory table

Pressing the button Save will save all changes made on the in-memory table and reset the History. A confirmation dialog box is presented before this action.

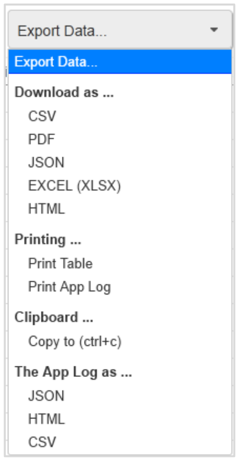
Saving All Changes applies it on the following sequence:

1. **Deleting Rows**
2. **Inserting / Adding Rows**
3. **All Column** changes, as that will changes all rows respecting the defined filtering (VA report and/or table column header filters) and will may affect added rows respecting the filtering
4. **Individual Cell -** changes as those might, eventually overwrite some values on the 'All Column' update. Meaning, by cell changes will take priority over the mass changes

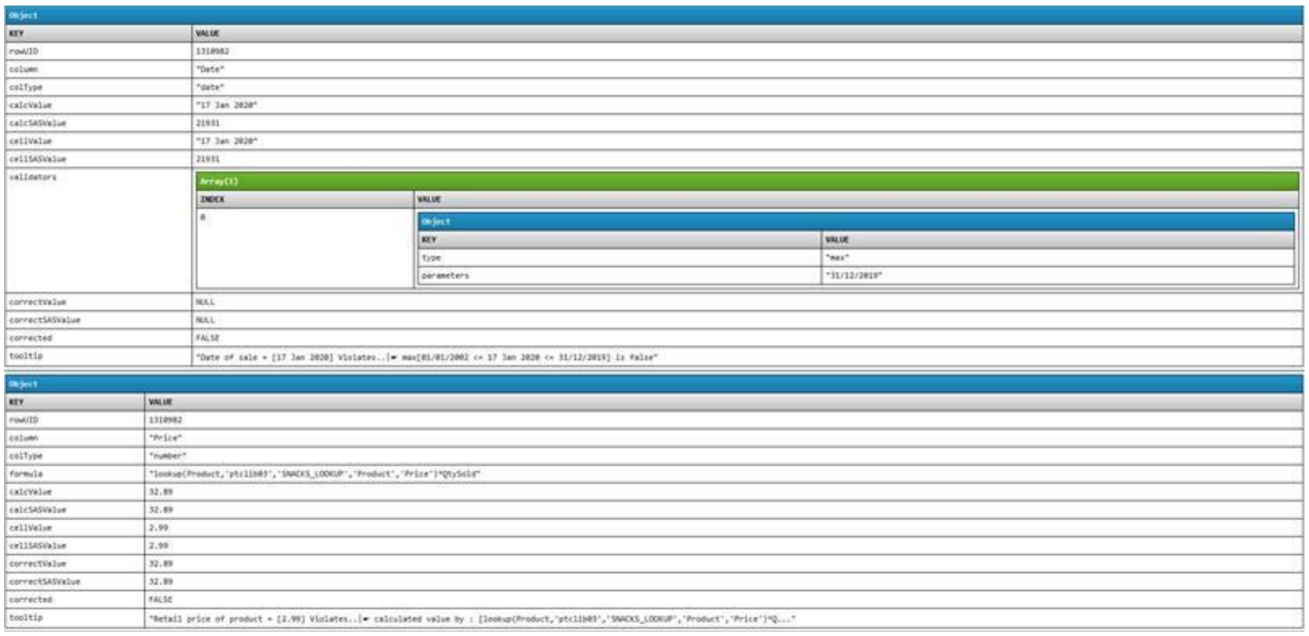
Persist changes on the physical (in disk) table

Pressing the button Save on Disk will save (all changes made in) the in-memory table on disk, so mean that in case of in-memory unload, it will be possible to recover all the meanwhile changes made.

Exporting Table Data



- If the respective module is active, a menu is added on the top of the Table area.
- Each export options are dependent on respective activation.
- Printing the table will use the same CSS used to present the data on the screen.
- To make sure that all options work properly, make sure that VA's configuration was changed as per described on [Configuring SAS VA iFrame parameters]()
- Exporting the App log will produce a file containing the browser console log until the moment of the export action, can be saved on the client device.
- Exporting the information about the Invalid Cells, can be found after the options for the **App Log**. It's possible to export as a JSON file, or as pretty HTML as like picture below:



Object

KEY	VALUE
rowUID	1310982
column	"Date"
colType	"date"
calcValue	"17 Jan 2020"
calcSASValue	21931
cellValue	"17 Jan 2020"
cellSASValue	21931
validators	Array(1)
correctValue	NULL
correctSASValue	NULL
corrected	FALSE
tooltip	"Date of sale = [17 Jan 2020] Violates..[➡ max[01/01/2002 <= 17 Jan 2020 <= 31/12/2019] is false"

Array(1)

INDEX	VALUE
0	Object

Object

KEY	VALUE
type	"max"
parameters	"31/12/2019"

Object

KEY	VALUE
rowUID	1310982
column	"Price"
colType	"number"
formula	"lookup(Product,'ptclib03','SNACKS_LOOKUP','Product','Price')*QtySold"
calcValue	32.89
calcSASValue	32.89
cellValue	2.99
cellSASValue	2.99
correctValue	32.89
correctSASValue	32.89
corrected	FALSE
tooltip	"Retail price of product = [2.99] Violates..[➡ calculated value by : [lookup(Product,'ptclib03','SNACKS_LOOKUP','Product','Price')*Q..."

Importing Table Data from Clipboard

It's possible to **Update** the table by "importing" / pasting a table from the clipboard using **'Ctrl+v'**. The pasted content MUST have the same structure, namely a header row with **exactly the same columns labels** (names).

- **USE WITH CAREFUL!!**

- Paste only rows having modifications or new. Rows which do not have any modification are also parsed and each cell content verified against the existing table, so avoid that as only takes time 😉, for example having a date column in which the incoming date format isn't equal to the one on the VA DDC Table.
- Point and click with the cursor in any cell of the table (for the Table object gaining the browser focus on it), then press **CTRL+V**.

1. **To add new rows**, the content of the column with the Unique Row ID must contain a value that does not exist, for example *00000000* or *99999999* . That ensures the row can't be found on the table and therefore is considered a new row. Being a new row, the value passed as Unique Row ID value, will not be used as the solution will automatically determine the next Row Unique ID.
2. If a column contains dates in a different format the one defined on the VA Report for that column, the solution will try to convert to the VA DDC object format. However, it's recommended to keep the same date format as shown on VA DDC object.
3. All cell changes are validated as if the change was made manually. However if default values were defined on the Control File **Check 'Setting Up the Control File' / validation / default**
   a. For the Numeric columns (floats and integers), this means overall length including precision (decimals) is tested. If the **min** and/or **max** were set on the Control File, that, is also validated, thus in case of violation the default value is used.
   b. For the String columns, if **minLength** and/or **maxLength** were set on the Control File, that, is also validated, thus in case of violation the default value is used.
   c. For the Date columns, if **default** were set on the Control File, that, is also validated, thus in case of violation the default value is used. If not defined a default, the current date is used.
4. All changes on the table will be recorded on the History module and therefore Undo is possible.

Detailed App log

Basically, all the actions executed by the component are logged on the Browser console log, which can be exported to a JSON, HTML or CSV file. The exported file, containing the log until the moment of the export action, can be saved on the client device.

APPENDIX 1 – ADDITIONAL FUNCTIONS

Header Filtering functions

A space **must exist** after the function and before the condition, e.g.: ***>= 1800***

Function	Input Argument(s)	Return value
> 'value to search'	Greater than..	All values greater than 'value to search'
>= 'value to search'	Greater or equal than..	All values greater or equal than 'value to search'
< 'value to search'	Smaller than..	All values smaller than 'value to search'
<= 'value to search'	Smaller or equal than..	All values smaller or equal than 'value to search'
= 'value to search'	equal to .. (is the default for numeric columns, so isn't needed, 'value to search' is enough)	All values equal to 'value to search'
!= 'value to search'	Not equal to ..	All values not equal to 'value to search'
like 'value to search'	like to .. (is the default for alphanumeric / string columns, so isn't needed, 'value to search' is enough)	All values alike to 'value to search'
in ['values to search'**]**	values separated by comma. If string, those values MUST be enclosed by quotes	Returns all rows where column value is on the [ ]
!in ['values to search'**]**	values separated by comma. If string, those values MUST be enclosed by quotes	Returns all rows where column value is **not** on the [ ]
btw ['value 1' , 'value 2'**]**	**between** values separated by comma. If string, those values MUST be enclosed by quotes	Returns all rows where column value is on **between** ['value 1' and 'value 2']
!btw ['value 1' , 'value 2'**]**	values separated by comma. If string, those values MUST be enclosed by quotes	Returns all rows where column value is **not between** ['value 1' and 'value 2']
out ['value 1' , 'value 2'**]**	values separated by comma. If string, those values MUST be enclosed by quotes	Returns all rows where column value is **not between** ['value 1' and 'value 2']
<!> 1 | 0	**Invalid Cells** Boolean value! 1 or 0	**1** = Returns all rows where column has **invalid cell** **0** = Returns all rows where column has **valid cell**

Aggregation functions

Function	Description
mad	Compute the median absolute deviation.
max	Compute the maximum value.
avg mean	Compute the mean value.
median	Compute the median.
min	Compute the minimum.
mode	Computes the mode.
prod	Compute the product.
count	Compute the number of occurrences.
countUnique	Compute the unique values in a column
std	Compute the standard deviation.
sum	Compute the sum.
var variance	Compute the variance.
sterr (stdErr, stderr)	returns the value of the standard error of the mean
gMean	returns the Geometric Mean that applies only to positive numbers
hMean	returns the Harmonic Mean of 'n' positive real numbers
zScore	returns the zScore, value's relationship to the mean of a group of values

Date and Time functions

Function	Input Argument(s)	Return value
date getSASDate	(<column label | column name>) *Must be a column type 'date' or 'datetime'*	The date in SAS format (number of days since 1Jan1960). If the receiving cell has a time format associated, that will be respected.
month getMonth	(<column label | column name>) *Must be a column type 'date' or 'datetime'*	the digit(s) of the month part
week getWeek	(<column label | column name>) *Must be a column type 'date' or 'datetime'*	the two digits of the week of year
day getDay	(<column label | column name>) *Must be a column type 'date' or 'datetime'*	the digit(s) of the day part
time getTime	(<column label | column name>) *Must be a column type 'time' or 'datetime'*	the time in seconds. If the receiving cell has a time format associated, that will be respected.
hours getHours	(<column label | column name>) *Must be a column type 'time' or 'datetime'*	the hour digit(s)
minutes getMinutes	(<column label | column name>) *Must be a column type 'time' or 'datetime'*	the minutes digit(s)
seconds getSeconds	(<column label | column name>) *Must be a column type 'time' or 'datetime'*	the second digit(s)

Other Functions

Function	Input Argument(s)	Return value
lookup	(<column label | column name providing the value to lookup>, <'casLib'>, <'casTable>' <'columnToLookup'>, <'returningValueColumn'>) *Column type providing the value to lookup could be of any kind. If a numeric value is returned, then it's possible to combine this function with other functions and/or operators.*	The found value on 'returningValueColumn' where content of 1st argument matches a value on 'columnToLookup' on the 'casTable' on 'casLib'. If no value found, returns **null.** If more than a value was found, returns the **first.**
getReportUserId	()	Returns the userId of whom is using the report
getReportUserName	()	Returns the username of whom is using the report
getReportUID	()	The report unique identifier
getReportName	()	The report name
getReportSessionUID	()	The report session unique identifier
getReportSessionName	()	The report session Name
getColumnValue	(<Column name or label>) *The row is the one where the cell is*	Cell **SAS Value**
getCurrentDate	()	Returns a localised string with current date
gerCurrentDateSAS	()	Returns SAS value for current date
getCurrentDateISO	()	Returns a string with a date on ISO format
getCurrentDateUTC	()	Returns a string with a date on UTC format
getTableFilters	()	Returns a string(fyied) JSON object containing the VA Filters and the Table header filters.
getAllColumnValues	(<Column name or label>)	Returns a JSON array of objects {<rowUID> : <cell value> }
getTableData	(<range>) where range can be one of: "visible", **"active"**, "selected", "all" "active" is the default	Returns a JSON array of objects (one per row) { <column 1> : <cell value>, <column 2 > : <cell value>, (…) }, (…)
getTableChangedRowsByUID	()	Returns a JSON array (one entry per row) [<row_uid>, (…)] Sorted by Row UID
getTableChangedRowsByPos	()	Returns a JSON array (one entry per row) [<row position on the table>, (…)]
getCellValue	(<row_uid>, <column name | label>)	Returns the cell value on the intersection of the provided Row_UID and Column. If not exist / invalid, returns NULL.

String functions

First parameter should be always a visible table column (name or label) or a string enclosed by “”

between(<column name|label or string>, left, right)

Extracts a string between `left` and `right` strings.

camelize(<column name|label or string>)

Remove any underscores or dashes and convert a string into camel casing.

capitalize(<column name|label or string>)

Capitalizes the first character of a string.

chompLeft(<column name|label or string>, prefix)

Removes `prefix` from start of string.

chompRight(<column name|label or string>,suffix)

Removes `suffix` from end of string.

collapseWhitespace(<column name|label or string>)

Converts all adjacent whitespace characters to a single space, also removes new-line, tab and other spacing type characters.

contains(<column name|label or string>, ss)

Returns true if the string contains `ss`.

Alias: **include (<column name|label or string>, ss)**

count(<column name|label or string>, substring)

Returns the count of the number of occurrences of the `substring`.

dasherize(<column name|label or string>)

Returns a converted camel cased string into a string delimited by dashes.

decodeHTMLEntities(<column name|label or string>)

Decodes HTML entities (like &amp, &lt, …) into their string representation.

deLatinise(<column name|label or string>)

Removes accents from Latin characters.

dotIt(<column name|label or string>)

Returns converted camel cased string into a string delimited by dots.

escapeHTML(<column name|label or string>)

Escapes the html, for example: '<div>hi</div>' will become ‘&lt;div&gt;hi&lt;/div&gt’

ensureLeft(<column name|label or string>, prefix)

Ensures string starts with `prefix`.

ensureRight(<column name|label or string>, suffix)

Ensures string ends with `suffix`.

humanize(<column name|label or string>)

Transforms the input into a human friendly form.

include(<column name|label or string>, ss)

Alias of function **contains**.

left(<column name|label or string>, n)

Return the substring denoted by `n` positive left-most characters.

length(<column name|label or string>)

Property to return the length of the string.

pad(<column name|label or string>, len, [char])

Pads the string in the centre with specified character. `char` may be a string; default is a space.

padLeft(<column name|label or string>, len, [char])

Pads the string in the left with specified character. `char` may be a string; default is a space.

padRight(<column name|label or string>, len, [char])

Pads the string in the right with specified character. `char` may be a string; default is a space.

repeat(<column name|label or string>, n)

Returns a string repeated `n` times.

Alias : **times(<column name|label or string>, n)**

replaceAll(<column name|label or string>, ss, newstr)

Return the new string with all occurrences of `ss` replaced with `newstr`.

right(<column name|label or string>, n)

Return the substring denoted by `n` positive right-most characters.

setValue(<column name|label or string>, value)

Sets the string to a `value`.

slugify(<column name|label or string>)

Converts the text into a valid url slug. Removes accents from Latin characters.

splitLeft(<column name|label or string>, sep, [maxSplit = -1, [limit]])

Returns a comma separated string. Split from the left at `sep`. Performs at most `maxSplit` splits and slices the result into an array with at most `limit` elements.

splitRight(<column name|label or string>, sep, [maxSplit = -1, [limit]])

Returns a comma separated string. Split from the left at `sep`. Performs at most `maxSplit` splits and slices the result into an array with at most `limit` elements.

strip(<column name|label or string>, [string1],[string2],...)

Returns a new string with all occurrences of `[string1],[string2],...` removed.

stripLeft(<column name|label or string>, [chars])

Returns a new string in which all chars have been stripped from the beginning of the string (default whitespace characters).

stripRight(<column name|label or string>, [chars])

Returns a new string in which all chars have been stripped from the end of the string (default whitespace characters).

stripPunctuation(<column name|label or string>)

Strips all the punctuation (character used to punctuate / separate letters and/or words.

stripTags(<column name|label or string> , [tag1],[tag2],...)

Strip all the HTML tags or tags specified by the parameters.

times(<column name|label or string>, n)

Alias for repeat.

titleCase(<column name|label or string>)

Returns a string with the first letter of each word uppercased, including hyphenated words

toFloat(<column name|label or string>, [precision])

Return the float value.

toInt(<column name|label or string>) / toInteger(<column name|label or string>)

Return the number value in integer form. Can also parse hex values (e.g.: ‘0xff’ => 255)

toString(<column name|label or string>)

Return the string representation of number passed as input.

trim(<column name|label or string>)

Return the string with leading and trailing whitespace removed.

trimLeft(<column name|label or string>)

Return the string with leading and whitespace removed

trimRight(<column name|label or string>)

Return the string with trailing whitespace removed.

truncate(<column name|label or string>, length, [chars])

Truncates the string, accounting for word placement and character count.

underscore(<column name|label or string>)

Returns converted camel cased string into a string delimited by underscores.

unescapeHTML(<column name|label or string>)

Unescapes the HTML, for example: '&lt;div&gt;hi&lt;/div&gt;' => ‘<div>hi</div>’

wrapHTML(<column name|label or string>)

wrapHTML helps to avoid concatenation of element with string. the string will be wrapped with HTML Element and their attributes.

Examples:

wrapHTML('Joao') → ‘<span>Joao</span>’

wrapHTML('Joao', 'div') → ‘<div>Joao</div>’

wrapHTML('Joao', 'div', ‘{"class": "left bullet"}’) → ‘<div class="left bullet">Joao</div>’

wrapHTML('Joao', 'div', ‘{"id": "content", "class": "left bullet"}’) → ‘<div id="content" class="left bullet">Joao</div>’