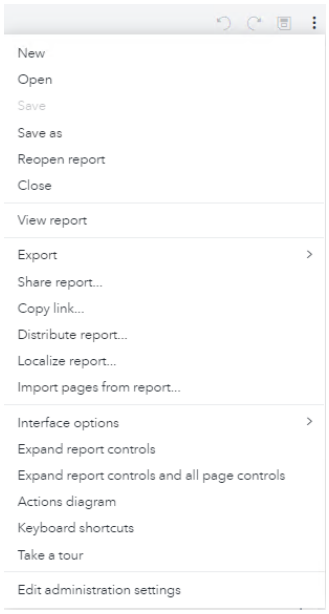
User Access Matrix – Column and Row Level Security for CAS Tables Documentation

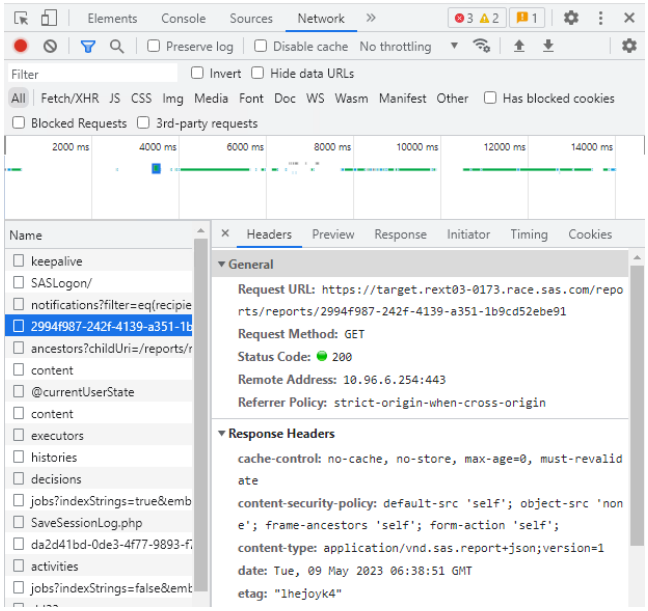
Product: Information Catalog
Version: SAS Viya 4
Date: 12/04/2023

Setting up the VA Table Editor

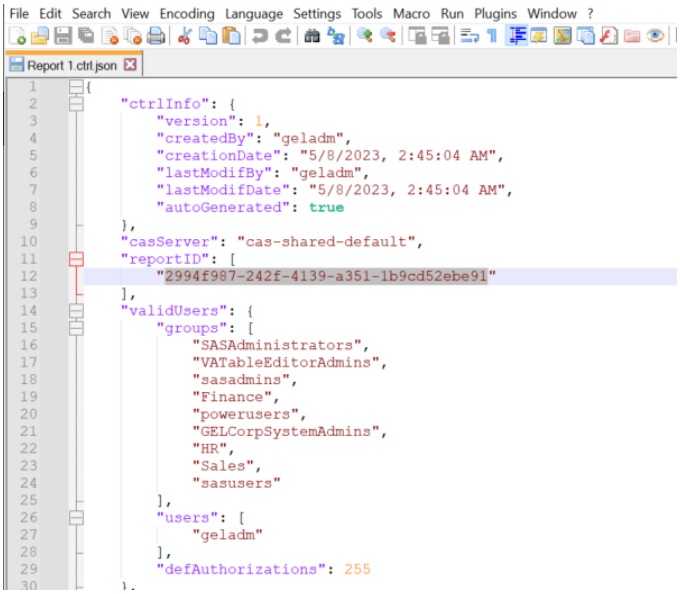
1. Unzip **va-table-editor-master.zip** to **D:**
2. Navigate to **D:\va-table-editor-master\App** and select **VATableEditor_DDC.html**
3. Scroll to the OnLoad and OnLoad functions and edit the ctrlFileLocation
4.
```
window.onload = function ()
{
    const VATE = new VATableEditor(
                                { // 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,
                                    CtrlFileLocation: "./",
```
5. Go to SAS Visual Analytics and create a new report
6. Click **Objects** and select **Data-driven content**
7. Save the report
8. Retrieve the report id
    a. Press **Ctrl + Shift + I**
    b. Click the arrow followed by **Network**
    c. Click on the clear button 
    d. Then, click on the more button followed by the option **Reopen report**



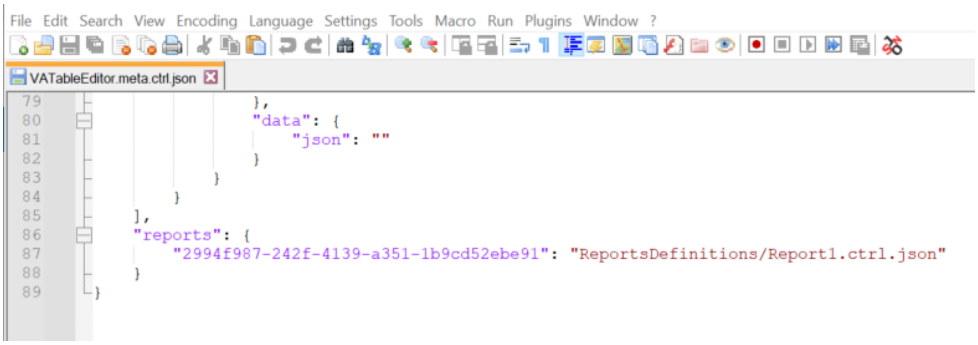
e. Find the Request URL containing the Report ID (i.e., **https://target.rext03-0173.race.sas.com/reports/reports/2994f987-242f-4139-a351-1b9cd52ebe91**, where **2994f987-242f-4139-a351-1b9cd52ebe91** is the report id)



9. Navigate to **D:\va-table-editor-master.zip \App\ReportDefinitions**, and click on **Report 1 ctrl.json** to edit in Notepad++
10. Change the report ID in the json file to the report ID of the report, rename the file to the name of the report saved



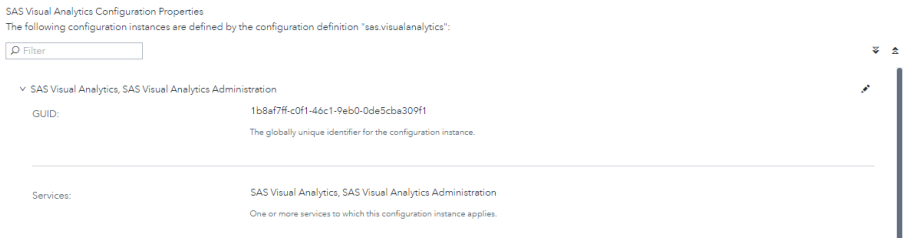
11. Navigate to **D:\va-table-editor-master.zip\App** and click on **VATableEditor.meta.ctrl.json**
12. Scroll to the bottom of the json file and edit the **reports** field to add the report ID of the saved report and the path



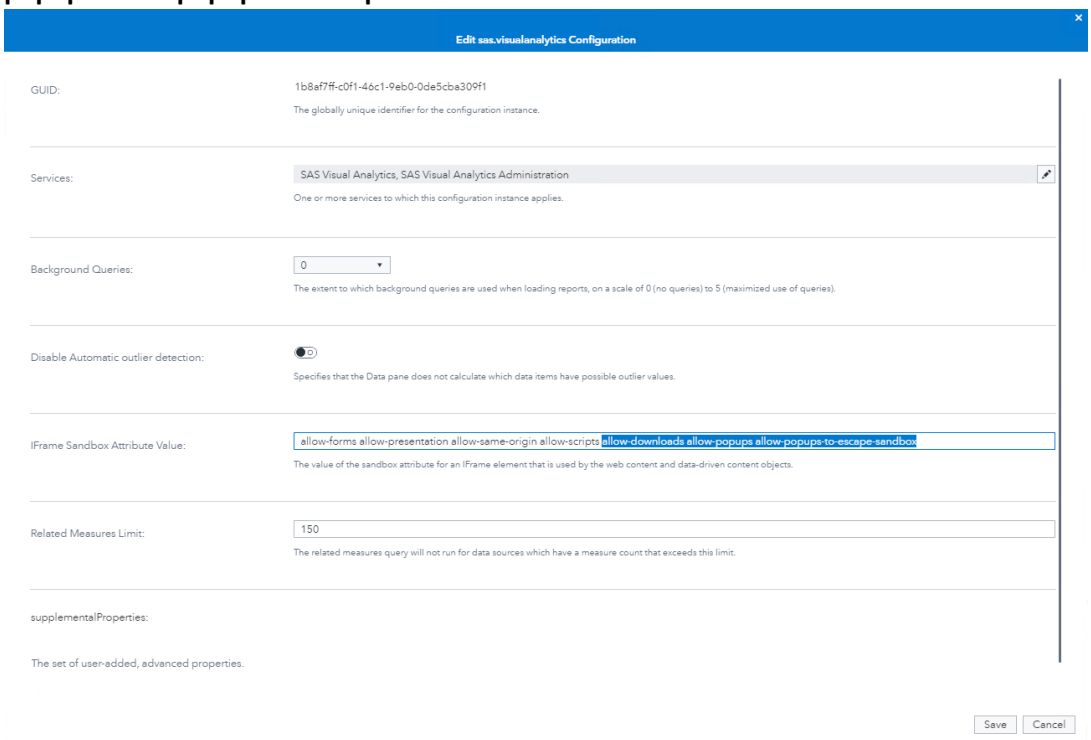
13. Zip the edited **va-table-editor-master** folder
14. In this section, we will be setting up VA Table Editor for the interface to view and edit the specific column and row level access permissions.
    i. Using MobaXterm, upload **va-table-editor-master.zip** to **/home/cloud-user/**
    ii. Switch to root using
    ```
    sudo su
    ```
    iii. Unzip **va-table-editor-master.zip**
    ```
    unzip va-table-editor-master.zip
    ```
    iv. Change the current directory to **va-table-editor-master/Viya4/**
    ```
    cd va-table-editor-master/Viya4/
    ```
    v. Run **install_vate.sh** to install VA Table Editor
    ```
    ./install_vate.sh
    ```
    vi. Run **register_client.sh** to register the client
    ```
    ./register_client.sh
    ```
    Note that if you encounter this when running **install_vate.sh** and/or **register_client.sh**:
    ```
    [root@pdcesx03204 Viya4]# ./install_vate.sh
    bash: ./install_vate.sh: Permission denied
    ```
    To change the permissions of **install_vate.sh** and/or **register_client.sh**

```
chmod 777 ./install_vate.sh
chmod 777 ./register_client.sh
```

vii. Login to SAS Environment Manager. Go to **Configuration**, under the View dropdown, select **Definitions**. Search for **sas.visualanalytics**, and edit **SAS Visual Analytics, SAS Visual Analytics Administration**.



viii. In the pop-up, under **IFrame Sandbox Attribute Value**, add "**allow-downloads allow-popups allow-popups-to-escape-sandbox**" and click **save**.



15. Go to SAS Visual Analytics and replace **https://BASEURL/SASVisualAnalytics/resources/custom_table.html** to **https://BASEURL/vate/App/VATableEditor_DDC.html?PageSize=100** (i.e., BASEURL = target.rext03-0173.race.sas.com)

Setting up the Job Executions

1. Create a new job execution called '*Upload UAM*'
2. In this section the process and the logic behind the SAS codes of **Upload UAM** will be explained
   a. First, we initialize the CAS session and assign all available CAS libraries to allow access to all libraries. The data step then code checks if the uploaded file is a CSV file and displays an error message in HTML format if it is not. We also define a file reference named '**WLIST**' and specify the file path and name of the uploaded CSV file.

```
cas;
caslib _all_ assign;

data _null_;
length filename $1024;
filename = htmlencode(strip("&_WEBIN_FILENAME"));
call symputx('_WEBIN_FILENAME', filename);
if (upcase("&_WEBIN_FILEEXT") ne 'CSV') then do;
  rc = dosubl('ods _all_ close;');
  file _webout;
  put '<!DOCTYPE html>';
  put '<html>';
  put '<head><title>Program Error</title></head>';
  put '<body>';
  put '<h1>ERROR: Uploaded file "' filename +(-1) '" is not a CSV file.</h1>';
  put '</body>';
  put '</html>';
  abort cancel;
end;
run;
S
/* Create a FILEREF for the uploaded file */
filename WLIST filesrvc parenturi="&SYS_JES_JOB_URI"
  name="&_WEBIN_FILENAME"
  contenttype="&_WEBIN_CONTENT_TYPE";
```

   b. The default rules for variable and dataset naming are changed to allow any characters and extends the length of dataset names, providing more flexibility in naming variables and datasets. After, e delete the existing promoted dataset in the caslib called **column_row_level**, which we will replace shortly. The **PROC IMPORT** step will then import the csv file into the CAS table **column_row_level** located in the library **CASUSER.**

```
/* Set options to support non-SAS name */
options validvarname=any validmemname=extend;

proc datasets library=public nolist;
   delete column_row_level;
run;
S
/* Import the uploaded CSV file */
proc import datafile=WLIST
  out=casuser.column_row_level
  dbms=csv
  replace;
  getnames=yes;
run; quit;
```

   c. The data step creates the dataset **column_row_level** in the public library from the dataset **casuser.column_row_level**, the **caslib** column is renamed, and a unique ID is generated based on the values of other variables. The next data step will then generate HTML code to the output of the job execution to display a success message indicating that a file has been uploaded and prompts the user to reopen the report.

```
data public.column_row_level (promote=yes);
set casuser.column_row_level;
rename 'caslib'n=caslib;
Unique_ID= 1/2 * (put(_threadid_,8.) + Put(_n_,8.)) * ((put(_threadid_,8.) + Put(_n_,8.)) + 1 ) + Put(_n_,8.);
run;

data _null_;
file _webout;
put '<!DOCTYPE html>';
put '<html>';
put '<head><title>Success</title></head>';
put '<body>';S
put "<h1>The file has been uploaded. Please reopen the report.</h1>";
put '</body>';
put '</html>';
```

3. Right click the **Upload UAM** job, select **Edit**, then **HTML form**
4. In this section the process and the logic behind the HTML form of **Upload UAM** will be explained
   a. The head section of the code contains the title of the page, along with a JavaScript function **ShowLoading** that is triggered when the form is submitted. The function displays a "Loading..." message and hides the input fields and labels while the file is being uploaded and processed.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Upload UAM File</title>
    <script type="text/javascript">
      function ShowLoading(e) {
      var column2 = document.getElementById('column2');
      var content = document.createElement('div');
      content.innerHTML = 'Loading...';
        if (column2.firstChild) {
          column2.insertBefore(content, column2.firstChild);
        } else {
          column2.appendChild(content);
        }
            var inputs = document.querySelectorAll('input, label');
          inputs.forEach(function(element) {
            element.style.display = 'none';
          });
      }
    </script>
  </head>
```

   b. The body section of the code contains the HTML form used for file uploading. It includes a file input field where the user can select the CSV file to upload. Upon submitting the form, the **ShowLoading** function is triggered.

```
  </head>
  <body role="main" class="jobexec_body">
    <form class="jobexec_form" action="/SASJobExecution/" method="post" target="_self"
      enctype="multipart/form-data" runat="server"  onsubmit="showLoading()">
      <input type="hidden" name="_program"     value="$PROGRAM$">
      <input type="hidden" name="_action"      value="execute"/>
      <input type="hidden" name="_output_type" value="html"/>
      <input type="hidden" name="_csrf"        value="$CSRF$">
      <div style="display: flex;">
        <div style="flex: 1; text-align: center; width: 33.33%;">
          <label for="myfile">Choose a CSV file to upload:</label>
          <input type="file" name="myfile" id="myfile" required class="jobexec_sample_input_file" style="background-color: #4079b1; color: white; padding: 10px 20px; border: none; border-radius: 5px; cursor: pointer; width:90%;"/>
        </div>
        <div id='column2' style="flex: 1; text-align: center;" width: 33.33%;>
        <br>
        <input type="submit" target="_self" value="Upload UAM File" class="jobexec_sample_input_submit" style="background-color: #4079b1; color: white; padding: 12.25px 20px; border: none; border-radius: 5px; cursor: pointer; width: 90%;"/>
      </div>
      <div style="flex: 1;"></div>
      </div>
    </form>
  </body>
</html>
```

3. Create a new job execution called '*Run UAM*'
4. In this section the process and the logic behind the SAS codes of **Run UAM** will be explained
   a. First, we initialize the CAS session and assign all available CAS libraries to allow access to all libraries. Then we define the macro variable '**adminGroup**' and assign it to the value '**SASAdministrators**'. The default rules for variable and dataset naming are changed to allow any characters and extends the length of dataset names, providing more flexibility in naming variables and datasets. The DATA step creates dataset **column_level** by selecting rows from the **column_row_level** table where *Sec_level = 'C'.*

```
cas;
caslib _all_ assign;
%let adminGroup = SASAdministrators;
options validvarname=any validmemname=extend;

data column_level;
set public.column_row_level;
where Sec_level ="C";
run;
```

b. Following that, we define a macro program called **reset_col_level**, which loops through the columns. The PROC CAS step uses the **accessControl.updSomeAcsColumn** action to grant Read Info & Select permission. The purpose of this is to reset the access control settings for all columns, so that column level security initially set can be revoked.

```
%macro reset_col_level(table);
proc contents data=test.cfmast out=tabcol(keep=name) noprint;
            quit;

            proc sql noprint;
                select count(name),name
                    into :tabcnt, :tabnm1- from tabcol;
                drop table tabcol;
            quit;
        %do i=1 %to &tabcnt;
        proc cas;
            accessControl.updSomeAcsColumn
                acs={
                {caslib="test",
                table="&table",
                column="&&tabnm&i",
                identity="*",
                identityType="Group",
                permType="Grant",
                permission="ReadInfo"},
                {caslib="test",
                table="&table",
                column="&&tabnm&i",
                identity="*",
                identityType="Group",
                permType="Grant",
                permission="Select"}};
 run;
 %end;
%mend;
```

c. Next, we store the values and count of the **column_level** table as macro variables. We also create another dataset named **column_level_distinct** by selecting distinct columns and counts the occurrences of distinct combinations of the columns. Then, it stores the values of **column_level_distinct** into macro variables.

```
/* Create macro variables for each whitelist record */
proc sql noprint;
    select count(column),
        caslib,
        table,
        column,
        userGroup
    into :cnt
        , :caslib1-
        , :table1-
        , :column1-
        , :userGroup1-
    from column_level;
quit;

/* Distinct Table-Column List */
proc sql noprint;
    create table column_level_distinct as
        select distinct caslib,
            table,
            column,
            count(*) as dcnt
        from column_level;
quit;

proc sql noprint;
    select count(column),
        caslib,
        table,
        column
    into :d_cnt
        , :D_caslib1-
        , :D_table1-
        , :D_column1-
    from column_level_distinct;
quit;
```

d. Following that, we define a macro program called **looper_DenyALL** which loops through the distinct column-level rules. For each iteration, the macro variables are used to retrieve the caslib, table, and column values. The **PROC CAS** step uses the

accessControl.updSomeAcsColumn action to grant Read Info & Select permission to SASAdministrators and denying Read Info & Select permission to all other identities for the specified column.

```
/* Loop start - apply permission for each column */
%macro looper_DenyALL;
    %do i=1 %to &d_cnt;
        %put DENY ALL EXCEPT ADMIN | NEXT TO PROCESS:;
        %put caslib: &&caslib&i;
        %put table: &&table&i;
        %put column: &&column&i;

        proc cas;
            accessControl.updSomeAcsColumn /
                acs={
                {caslib="&&D_caslib&i",
                table="&&D_table&i",
                column="&&D_column&i",
                identity="*",
                identityType="Group",
                permType="Deny",
                permission="ReadInfo"},
                {caslib="&&D_caslib&i",
                table="&&D_table&i",
                column="&&D_column&i",
                identity="*",
                identityType="Group",
                permType="Deny",
                permission="Select"},
                {caslib="&&D_caslib&i",
                table="&&D_table&i",
                column="&&D_column&i",
                identity="&adminGroup",
                identityType="Group",
                permType="Grant",
                permission="ReadInfo"},
                {caslib="&&D_caslib&i",
                table="&&D_table&i",
                column="&&D_column&i",
                identity="&adminGroup",
                identityType="Group",
                permType="Grant",
                permission="Select"}
                };
        run;
        quit;
    %end;
%mend;
```

e. Subsequently, we define a macro program called **looper_GrantWhitelist** which loops through the column-level rules. For each iteration, the macro variables are used to retrieve the caslib, table, and column values. The **PROC CAS** step uses the **accessControl.updSomeAcsColumn** action to grant Read Info and Select permissions to the specified user groups for the specified columns.

```
%macro looper_GrantWhitelist;
    %do i=1 %to &cnt;
        %put GRANT WHITELIST | NEXT TO PROCESS:;
        %put caslib: &&caslib&i;
        %put table: &&table&i;
        %put column: &&column&i;
        %put user group: &&userGroup&i;

        proc cas;
            accessControl.updSomeAcsColumn /
                acs={
                {caslib="&&caslib&i",
                table="&&table&i",
                column="&&column&i",
                identity="&&userGroup&i",
                identityType="Group",
                permType="Grant",
                permission="ReadInfo"},
                {caslib="&&caslib&i",
                table="&&table&i",
                column="&&column&i",
                identity="&&userGroup&i",
                identityType="Group",
                permType="Grant",
                permission="Select"}
                };
        run;
        quit;
    %end;
%mend;
```

f. Finally**,** we define a macro program called **apply_rls** which loops through the row-level rules. A dataset called **row_level** is created where *Sec_level = 'R'*. A **PROC SQL** statement counts the number of rows in **row_level** and runs the subsequential codes if **row_level** is not

empty. The **DATA** step modifies the **row_level** dataset by creating a variable **grprls** which represents the filter condition for row-level security. The values of **row_level** is then stored as macro variables. Finally, The **PROC CAS** step uses the **accessControl.updSomeAcsTable** action to grant Read Info and Select permissions to the specified user groups based on the defined filter condition.

```
%macro apply_rls;
    data row_level;
        set public.column_row_level;
        where Sec_level ="R";
    run;

    Proc sql noprint;
        select count(*) into :st
            from row_level;
    quit;

    %if &st > 0 %then
        %do;
            proc contents data=row_level out=rlsd(where=(NAME contains
                "row") keep=name) noprint;
            quit;

            proc sql noprint;
                select count(name),name
                    into : rlsc , :rlnm separated by "," from rlsd;
                drop table rlsd;
            quit;

            data row_level;
                set row_level;
                %do i=1 %to &rlsc;
                    if row&i ne "" then
                        row_&i=catq('1A',row&i);
                    drop row&i;
                    rename row_&i=row&i;
                %end;
            run;

            data row_level;
                attrib grprls length=$32767.;
                set row_level;
                    grprls = cat(column, " ", &rlnm);
            run;

            proc sql noprint;
                select count(*),caslib,table,userGroup,grprls
                    into :cn, :cslm1-, :tbl- ,:uGp1-,:rls1-
                        from row_level;
            quit;

            %do i=1 %to &cn;
        %put GRANT ROW LEVEL SECURITY | NEXT TO PROCESS:;
        %put caslib: &&csln&i;
        %put table: &&tb&i;
        %put identity: &&uGp&i;
        %put filter: &&rls&i;

                proc cas;
                    accessControl.updSomeAcsTable /
                        acs={
                        {caslib="&&csln&i",
                        table="&&tb&i",
                        identity="&&uGp&i",
                        identityType="Group",
                        permType="Grant",
                        permission="Select",
                        filter="&&rls&i"},
                        {caslib="&&csln&i",
                        table="&&tb&i",
                        identity="&&uGp&i",
                        identityType="Group",
                        permType="Grant",
                        permission="ReadInfo"}};
                run;

            %end;
        %end;
%mend;
```

g. The next data step will then generate HTML code to the output of the job execution to display a success message indicating that row and column level security has been applied.

```
data _null_;
file _webout;
put '<!DOCTYPE html>';
put '<html>';
put '<head><title>Success</title></head>';
put '<body>';
put "<h1>Row and column level security has been applied</h1>";
put '</body>';
put '</html>';
run;
```

5. In this section the process and the logic behind the HTML form of **Run UAM** will be explained
   a. The head section of the code contains the title of the page, along with a JavaScript function **ShowLoading** that is triggered when the form is submitted. The function displays a "Loading..." message and hides the input fields and labels while the file is being uploaded and processed.

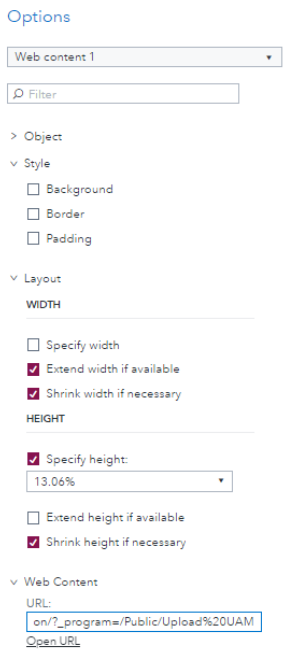
```
<!DOCTYPE html>
<html lang="en">
   <head>
      <title>Run UAM File</title>
      <link rel="stylesheet" href="/SASJobExecution/theme">
      <script type="text/javascript">
         function ShowLoading(e) {
         var column2 = document.getElementById('column2');
         var content = document.createElement('div');
         content.innerHTML = 'Loading...';
           if (column2.firstChild) {
             column2.insertBefore(content, column2.firstChild);
           } else {
             column2.appendChild(content);
           }
                var inputs = document.querySelectorAll('input, label');
               inputs.forEach(function(element) {
                 element.style.display = 'none';
               });
         }
      </script>
   </head>
```

   b. The body section of the code contains the HTML form used for running a UAM file. The form includes a submit button labelled “Run UAM”. Upon submitting the form, the **ShowLoading** function is triggered.

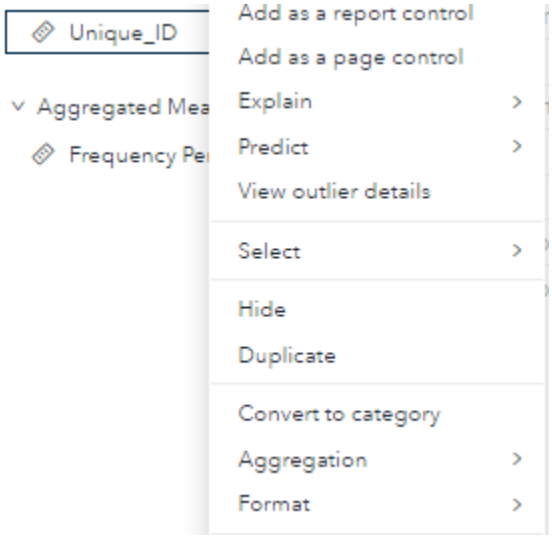
```
   <body role="main" class="jobexec_body">
      <form class="jobexec_form" action="/SASJobExecution/" method="post" target="_self"
          enctype="multipart/form-data" runat="server"  onsubmit="ShowLoading()">
         <input type="hidden" name="_program"     value="$PROGRAM$">
         <input type="hidden" name="_action"      value="execute"/>
         <input type="hidden" name="_output_type" value="html"/>
         <input type="hidden" name="_csrf"        value="$CSRF$">
         <div style="display: flex;">
            <div style="flex: 1; text-align: center;"></div>
            <div id="column2' style="flex: 1; text-align: center;">
               <input type="submit" target="_self" value="Run UAM" class="jobexec_sample_input_submit" style="background-color: #4079b1; color: white; padding: 12.25px 20px; border: none; border-radius: 5px; cursor: pointer; width: 90%;"/>
            </div>
            <div style="flex: 1;"></div>
         </div>
         </div>
      </form>
   </body>
</html>
```

Building the VA Report

1. In the report previously created, click **Objects,** and select **Web Content**
2. Ensure the web content object is placed above the Data Driven Content and drag the web content until it is the smallest height.
3. Click **Options**, and under **Web Content** insert the **Upload UAM** job execution URL to the URL

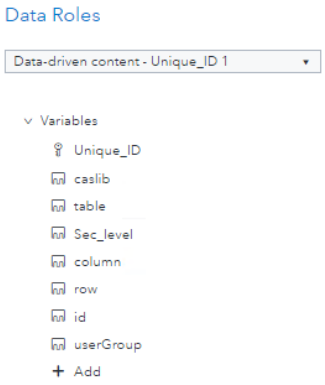


4. Upload the UAM csv file using the Upload UAM job in the report
5. Once the file has been uploaded. Restart the report and click **Data**
6. Add **public.column_row_level** to the report
7. Right click on **Unique_ID** and select **Convert to category**



8. Once **Unique_ID** is converted into a Category, right click on **Unique_ID** again and select **Set as unique row identifier**

9. On the Data Driven Content object, click **Assign Data**, and add all variables except **Frequency** and **Frequency Percent**. Click **OK.**
10. Click Roles and drag rearrange the variable such that they are in this order.



11. Click Options, and under Web Content, replace the URL to **https://BASEURL /vate/App/VATableEditor_DDC.html?PageSize=100** (i.e., BASEURL = target.rext03-0173.race.sas.com)
12. Go back to **Options** and select **Web Content**
13. Ensure the new web content object is placed below the Data Driven Content and drag the web content until it is the smallest height.
14. Click **Options**, and under Web Content insert the Run UAM job execution URL to the URL
15. The final report has the following layout