

Customizations to the SAS Multidimensional Report Viewer

Karl Schaefer, Qualex Consulting Services, Inc., Meherrin, VA
Steve Wagar, Qualex Consulting Services, Inc., Trenton, GA
Sarah Mitchell, Qualex Consulting Services, Inc., Yakima, WA
Alan Leach, Qualex Consulting Services, Inc., Norwood, OH

ABSTRACT

The SAS Multidimensional Report Viewer (MRV) is a web-based tool that provides a way to create reports from multidimensional data bases (MDDBs). It is great out-of-the-box solution that allows users to quickly provide web-reports to an organization. However, we had client needs that required us to go inside the MRV, to dramatically change its appearance and functionality. The ability to customize the MRV through object-oriented techniques has been described by previous authors. In this paper we present specific customizations to the MRV that we made for various clients. First we describe simple external changes made that alter the general appearance of the MRV. Next, we describe how, using method overrides and instance variables, to make further customizations. For example, we construct syntactically correct WHERE clauses based on the current status of the MRV and show how this can be used to query SAS tables. Also, we describe how to override the output of the MRV in order to display additional information from extra titles to complete new tables using simple ODS (Output Delivery System) processing. In addition, we also offer a simple print solution using SAS ODS RTF output to circumvent the web browser printing limitations. Finally, we demonstrate the methods overrides needed in order to use the MRV using pool services with sessions.

INTRODUCTION

This paper will discuss some specific customizations that were done on the SAS Multidimensional Report Viewer (MRV) in order to meet reporting requirements. We will first discuss some of the specific requirements that we had of the MRV that forced us to customize. We will then illustrate how we met those requirements by discussing some general techniques and providing specific examples. We will attempt to provide the code for these specific examples. However, many of method overrides that we discuss are rather lengthy. Thus, anyone interested in receiving the code should contact the authors who can supply the code needed to complete all the customizations mentioned in this article.

BACKGROUND

The SAS MRV displays information in pre-generated pre-registered SAS Multi-Dimensional Databases (MDDBs) via web browsers. It is part of the SAS IntraNet® suite of products and works as an application dispatcher program. The MRV accepts name/value pairs via a SAS/IntraNet Service CGI program, processes the request, and outputs an HTML stream to the requesting browser. (See the SAS online documentation listed at the end of this paper for complete documentation on the MRV and SAS/IntraNet®.) The resulting report provides functionality via JavaScript that allows the user to drill down on the data, change views, etc. Colin Harris provided a good comprehension description and review of the MRV in his SUGI 26 paper entitled "Dynamic and Ad-hoc Reporting via the Web: Real World Comparison of the MDDB Report Viewer and AppDev Studio®".

By default the user generates a report by first going to the Report Layout Page. There the user selects the analysis variable(s), the down and across dimensions, statistic, etc.

The SAS MRV is coded using object-oriented techniques. The parent class is the SASHELP.WEBEIS™. Katz and Rood (2002) described the basics and the challenges of customizing the SAS MRV in their SUGI 27 paper entitled "Web-enabling a

Client/Server OLAP Application Using SAS/INTRNET Software's MDDB Report Viewer". Using these techniques to create our own subclass and by changing set-up parameters we significantly customized the MRV to change its functionality and appearance.

OUR REQUIREMENTS

Our clients wanted to use the MRV because of its drill down capabilities and quick deployment. However, they had very specific requirements for their reports that needed specialized customizations:

- The users wanted to streamline the reports as much as possible. For example, they wanted to go directly to an MRV via a single click from menu system. (i.e., they wanted to bypass the Report Layout Page). Also they did not need a lot of the MRV features such as Rotate and Layout Button, hierarchy and statistic selection, etc.
- The clients needed to change the general appearance of the reports by changing colors and adding custom titles and removing redundant labeling.
- The tables that they wanted to display were very wide, requiring scrolling to the right to see the 'Total' column by default. Thus, they needed the 'Total' column to the left side of the table.
- These wide tables caused the hardcopy printed from the browser to be truncated. Thus, the client needed a way to print the entire report.
- In addition to the MRV, the client needed additional small tables to display below the MRV that summarized other aspects of the data. These tables while using the base data for MDDB as the source data, had to reflect the same data state in the MRV. Thus, they had to apply any subsetting requested by the user via the MRV.
- The application would be running on a pool service using sessions, thus, we needed to modify the MRV to carry along the _PORT, and _SERVER, _SESSIONID name/value pairs throughout every drill down.

Here is a 'Before' shot using the default MRV:

Month	APR02	MAY02	JUN02	JUL02	AUG02	SEP02	OCT02	NOV02	DEC02
Drug Indication Group	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value
Disposition	\$440,652	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Discontinue	\$2,623,645	\$27,412	\$27,411	\$27,421	\$27,427	\$27,394	\$59,175	\$59,018	\$58,866
Hypertension	\$31,398,121	\$445,941	\$129,392	\$0	\$0	\$0	\$172,299	\$376,285	\$654,755
Lipid Lowering	\$61,868	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Pain Killers	\$149,082	\$39,312	\$0	\$0	\$0	\$0	\$0	\$0	\$0

Now the 'After' picture using our MRV customizations with some of the customizations highlighted:

Unnecessary Selection Buttons and Lists Removed

Custom Titles

Link to Printable Version

Total Column First

Change in Colors/Fon

Extra Table Querying Same Data as the MRV

Month	APR02	MAY02	JUN02	JUL02	AUG02	SEP02	OCT02	NOV02	DEC02
Drug Indication Group	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value	MTM Value
Disposition	\$440,652	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Discontinue	\$2,623,645	\$27,412	\$27,411	\$27,421	\$27,427	\$27,394	\$59,175	\$59,018	\$58,866
Hypertension	\$31,398,121	\$445,941	\$129,392	\$0	\$0	\$0	\$172,299	\$376,285	\$654,755
Lipid Lowering	\$61,868	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Pain Killers	\$149,082	\$39,312	\$0	\$0	\$0	\$0	\$0	\$0	\$0

THE CUSTOMIZATIONS

GOING DIRECTLY TO THE MRV

We initially used the MRV Report Layout Page to create a layout for our reports. However, our users did not want to have to recreate the report, as they typically wanted to see the same layout every day. They wanted to be able to click on a hyper link on a page and have a pre-set MRV report appear. Also, we needed to access the MRV from an already existing web menu created using SAS/IntrNet. Thus, we created SAS programs that generated a web page with a link to the URL needed to bring up the MRV.

There is nothing complex about the MRV requests - it is simply a set of name/value pairs sent to the web server like any other Common Gateway Interface (CGI)/application dispatcher program request. Due to the stateless nature of the HTTP and therefore SAS/IntrNet, numerous name/value pairs are required for a complete MRV request. We obtained the basic URL by cutting, pasting and modifying the value of the JavaScript variable 'bookmarkURL' that is generated by the MRV and found in the source HTML of the generated report. Once an MRV report is

generated, one can view the HTML source, copy the bookmarkURL value, and paste it into another document. This link will then contain all the name/value pairs to request the MRV report directly without having to go through the layout screen. (A complete description of these name value pairs can be found in the MRV documentation). Note though that the value from the source HTML needs to be modified slightly. It will actually reference a .csv filename. This needs to be removed and it will work properly. For example:

Link on Original MRV Report:

http://MYSERVER/cgi-bin/broker.exe/marketdata.csv?_service=MYSERVICE&_debug=0&.....other name/value pairs

Link in New HTML To Go Directly to MRV:

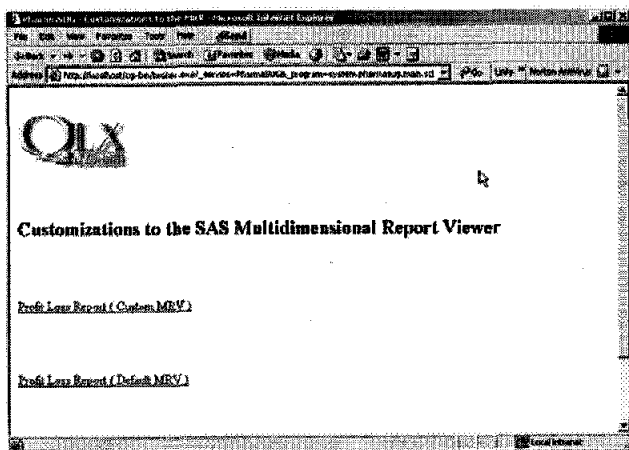
http://MYSERVER/cgi-bin/broker.exe?_service=MYSERVICE&_debug=0&.....other name/value pairs

With this URL there are a number of programmatic methods one can use to create the page that links directly to an MRV report. We simply created an application dispatcher program that generated an HTML page. In this program we created an anchor tag that would link to the report:

```
data _NULL_ ;
file _webout ;

put '<a href="http://" ;
put "& SERVER" ;
put '/cgi-bin/broker.exe?' ;
put '_service=PharmaSUGDEF' ;
put '&_debug=0' ;
put '&_program=sashelp.webeis.showrpt.scl' ;
put '&_mddb=SASDL.MARKETDATEDEF' ;
put '&_metabase=PHARMASUG' ;
put '&_D=Physical%20View' ;
put '&_AC=STARTDATE' ;
put '&_A=MTM_VALUE' ;
put '&_A1S1=SUM' ;
put '&_DC=1' ;
put '&_ACB=1' ;
put '&_ST=1' ;
put '&_GRT=' ;
put '&_GL=1' ;
put '&_GSC=1' ;
put '&_SSL=1' ;
put '&_SH=3' ;
put '&_SW=15' ;
put '&_GH=450' ;
put '&_GW=600' ;
put '&_NR=ALL' ;
put '&_BS=Block' ;
put '&_SV=BOOKDATE' ;
put '" target=" BLANK">' ;
put 'Profit Loss Report ( Default MRV )' ;
put '</a>' ;
```

Using the above and some additional code the resulting HTML page looked like this:



Any number of techniques can be used for the page that requests the MRV report. For example, one might use SAS to generate an HTML page containing JavaScript. The JavaScript might allow the user to select between reports, then generate the URL and fire the request to the server with all of the needed name/value pairs. Katz and Rood (2002) described such a screen in their work on MRV customizations.

MODIFYING ENVIRONMENT PARAMETERS

In addition to going directly to the MRV we wanted to eliminate some of the default features of the MRV that our users did not want. To accomplish this we used a simple program to set the global macros used by the MRV to values that we wanted. This program is referenced in the INIT option of the REQUEST statement in the PROC APPSRV of the SAS/IntrNet Service. The program file is defined to the service using an ALLOCATE FILE statement. For example, in our PROC APPSRV we added the following statements:

```
allocate file   srvmacro '... path to file ' ;
request init=srvmacro.programname.sas
```

The *programname* file contained the following statements:

```
%let _MRVSTAT=NO;          /* Turns off the
                             Statistic Selection */
%let _MRNOREACHTHRU=NO; /* Turns off Detail
                             Reachthrough */
%let _MRNODIMBOXES=NO; /* Turns off Down and
                             Across list boxes */
%let _MRTBLOC=5;          /* Turns off the
                             ToolBar*/
%let _MRNOFRAMES=NO;      /* Turns off Frames in
                             the Report Window */
```

A complete list the global macros variables used by the MRV can be found at in the MRV documentation.

USING A CUSTOM STYLE SHEET

By changing the cascading style sheet used by the MRV we were able to greatly affect the appearance of the report. By default the MRV uses a style sheet called default.css. Typically it is located in the 'webserver-root directory'/sasweb/IntrNet8/MRV/css' folder. We copied and renamed it create our own style sheet in which we changed various aspects of the styles. For example we changed the tdcell style class:

```
.tdcell { font-size: 8pt;
          text-align: right;
          font-family: Verdana, Arial,
          Helvetica, sans-serif;
          border-width: thin;
          font-weight: normal; border: none;
          background-color: c6cecd}
```

This gave all of the data cells a pleasing light blue background color. We also changed the color and background-colors attributes of the analycol style class to be the same. This eliminated the redundant headings on the analysis columns by making them seem invisible (they still exist in the HTML source).

These are just a couple examples of what can be done. If you want to know what style classes are associated with a specific part of the MRV report you can simply bring up a report then view the HTML source of the page.

Note that if you use a different style sheet you will need to change the value of the CSS name/value pair in the any initial report request URL.

TRAFFIC LIGHTING

We needed to display negative numbers in red in our report for quick reference by the users. To do this we created formats with embedded HTML. HTML 3.2 allows these specifications and the MRV is only HTML 3.2 compliant. HTML 4.0 would require modifications to use a cascading style sheet. Applying these formats when building the MDDB carried the traffic lighting through to the MRV reports. One such format we used is as follows:

```
/******
/* ($xxx,xxx) w/ HTML
/******
proc format;
  picture account
    low - < 0    = '000,000,000' </FONT>'
                  (prefix='<FONT COLOR=RED> ($')
                  = 'g'
                  ( prefix = '$' )
    0 < - high = '000,000,000'
                  (prefix='($')
                  = ' '
  ;
run;
```

SUBCLASSING THE MRV

All of the aforementioned customizations were done by changing things external to the MRV itself. We needed to make additional customizations that required us to subclass the MRV and override certain methods. Katz and Rood(2002) describe the general techniques for doing the subclassing and overriding. Note that when you create your own class you will need to change the CLASS name/value pair in any initial report request URL.

UTILITY METHODS

In subclassing we added new methods to facilitate later method overrides. One such method we created was the WRITE method, which encapsulated the SCL code needed to output _WEBOUT inside the MRV:

```
WRITE: /* WRITE (New method) */
```

```
/******
```

```

/* This method created to simplify the      */
/* repetitive steps taken to output text to */
/* the file ID for fileref _webout. Input   */
/* always has text to be put out, and then */
/* optional flag to not write out yet,      */
/* then final optional file ID to use       */
/* instead of default HTMLFILE_             */
/*****

```

```

method text_content $ 500

```

```

optional=
    do_not_write_yet 8
    output_file_id 8
;

```

```

if output_file_id in (0, .) then
output_file_id=htmlfile;
rc=fput(output_file_id,text_content);
if do_not_write_yet ne 1 then
rc=fappend(output_file_id);

```

```

endmethod;

```

This was used in numerous other method overrides to output text to the MRV report. For example, inside the overridden `_displayDefaultTitle` method we used this to output our custom titles.

OVERRIDING THE `_DISPLAYVALUES` METHOD

We used the `_displayValues` Method of the MRV class as a convenient method in which to create various SCL lists and variables that we would use in later methods. Typically we did this by querying various instance variables of the MRV. For example, we created a list of information about the current analysis variable that we used in later executing methods. To get this information we actually queried the model class instance variable of the MRV:

```

/*****
/* Create lists and fill them with the      */
/* information we need for the overridden    */
/* methods.                                 */
/*****

```

```

infolist = makelist();
emddbmid._getVarinfo(infolist);

```

The `EMDDBMID_` instance variable of the MRV class is an object and is the instance of the model class associated with the MRV viewer. The `_getVarInfo` method of this model class copies into the supplied list various sublists that contain information about the variables of the MDDb, i.e. MDDb metadata. Some of the sublists are named sublists for each of the variables in the MDDb. Each named sublist contains various named items with information about the variable such as the format('MDDBFMT') and type('MDDBTYPE'). For example, a partial CALL PUTLIST on the above INFOLIST might look like:

```

INFOLIST( DRUG_GROUP =(  VARNAME='DRUG_GROUP'
                        VARFMT='$30.'
                        VARINFMT='$30.'
                        VARLABEL='Drug
                        Indication Group'
                        VARLEN='30'
                        VARTYPE='C'
                        MDDBFMT='$30.'
                        MDDBTYPE='C'
                        ) [18965]
... )

```

Using these lists we can get information about the analysis variable that we can use in later methods. The `analvars_` instance variable of the MRV class contains the list of analysis

variables currently in the reports (for our requirements we only had one analysis variable per report). Here we create the `analvar_list` that we will use later:

```

/*****
/* Get the format of the Analysis Variables */
/*****

```

```

analvar_list = makelist();

```

```

analvar = getitemc ( analvars_ , 1 ) ;

```

```

/*      Note that here we are assuming only
one analysis variable */

```

```

analvar_fmt = getnitemc ( getniteml ( infolist,
analvar ) , "MDDBFMT" ) ;
analvar_lab = getnitemc ( getniteml ( infolist,
analvar ) , "VARLABEL" ) ;

```

```

rc = insertc ( analvar_list, analvar      , -1 ,
"NAME" ) ;

```

```

rc = insertc ( analvar_list, analvar_fmt , -1 ,
"FORMAT" ) ;

```

```

rc = insertc ( analvar_list, analvar_lab , -1 ,
"LABEL" ) ;

```

Once we have the information we want about the MRV and its variables we can tuck it away inside an instance variable of the MRV. Specifically we made it a named sublist, called `TABLE_LIST`, and inserted it into model object instance variable of the MRV. This will make it available to any of the other methods of the MRV class.

```

table_list = makelist();
rc=setniteml(table_list,analvar_list,
'ANALVARLIST',-1);
rc = insertl(emddbmid,table_list,
-1,'TABLE_LIST');

```

We employed this technique several times to create lists with information about the down variable, across variables, etc that we used in later methods.

MOVING THE 'TOTAL' COLUMN

A high priority requirement from our client was the ability to display the 'Total' column as the first column in the report. Their reports produced very wide tables so the user had to scroll all the way to the right, losing the ability to see the row headings. No facility is built into the standard MRV to perform this function, but we determined it could be done with just a few lines of code by manipulating one of the instance variables of the MRV.

The data to be displayed in the MRV output is contained in a list that the MRV creates during processing. The list is then processed sequentially during the execution of the `_displayAcrossCells` method, where the HTML output is written. By overriding that method we were able to manipulate the list of data to be displayed then call the parent method passing it our modified list.

Through iterative investigation and with the help of Technical Support from SAS the identifier for this list was found to be the first item in the argument list passed into the method `_displayAcrossCells`. Here is sample CALL PUTLIST output for this list:

```

Before ( ( DATE='APR02'
          _ANLSYS_='Measure Value'

```

```

    _STATS ='Sum'
    ) [17577]
  ( DATE='MAY02'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17579]
  ( DATE='JUN02'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17581]
  .
  .
  .
  ( DATE='JUL07'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17704]
  ( SDATE='TOTAL'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17705]
  ) [17207]

```

The custom code required to move the total column essentially rebuilds this list inserting the "Total" entry as the first entry in the list, and moving the other entries down one position in the list. This is accomplished by executing the following steps:

1. Create a new list.
2. Insert the last item from the original list as the first item in the new list
3. Sequentially process the remaining items in the original list, inserting them into the new list.
4. Replacing the original list in the "arglist" with the new list.
5. Executing the parent method to write out the values.

Here is the same list after manipulating it:

```

After ( ( DATE='TOTAL'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17577]
  ( DATE='APR02'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17579]
  ( DATE='MAY02'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17581]
  .
  .
  .
  ( DATE='JUN07'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17704]
  ( DATE='JUL07'
    _ANLSYS ='Measure Value'
    _STATS ='Sum'
    ) [17705]
  ) [17207]
  ) [16911]

```

Once we have the list in the order we want we call the parent method for the `_displayAcrossCells`.

CREATING A WHERE CLAUSE

The MRV uses MDDBs to display information. Typically these MDDBs are created from a SAS dataset or basetable. Our clients needed to display additional information from the basetable in the same page the as the MRV. Furthermore, the information in this table needed to reflect the current state of the data as reflected in

the MRV. Thus, if a user changed the filter value then data displayed in the extra table had to reflect the same subsetting by this filter.

To accomplish this we needed to make use of several of the instance variables of the MRV in order to build a WHERE clause that we could apply to a SAS Dataset. There are three ways that the user can change the subsetting status of the MRV:

- 1) By Selecting a new filter value
- 2) By drilling on a down value
- 3) By drilling on an across value

Recalling that HTTP and SAS/IntrNet are statelists we need to take into account each of these methods in order to get the complete current subsetting criteria with each request.

First we looked at the current filter values. These are contained in the `SUBSET_BY_` instance variable. It is an SCL list and contains the current filter criteria. For example, a CALL PUTLIST on `SUBSET_BY_` might show:

```

SUBSET_BY_ ( DATE=( ' Apr 4, 2002 '
                  ) [13279]
            ) [13289]

```

The `SUBSET_BY_` list will contain a sublist for each subsetting variable. Each sublist will contain the list of formatted values being currently being filtered for. Now the problem is that the values listed in the `SUBSET_BY_` list are the formatted values. In order to properly query the basetable, we need to know the format of the variable, in addition to the variable name and formatted value furnished by the `SUBSET_BY_` list. This information is obtained by querying the list returned by the `_getVarInfo` method of the `EMDDBMID_` object instance variable of the MRV class (see above). The list updated by this method contains information about all the variables in the MDDB. We found the sublist for the each variable in the `SUBSET_BY_` list, obtained the format and type, and used them to construct a series of WHERE clauses that we used to query the dataset. For example, for the above `SUBSET_BY_` list we would build a WHERE clause such as:

```
Where put( DATE,WORDDATE12.) = " Apr 4, 2002"
```

(Note that the WHERE clauses built in this manner may not make use of any indexing done on the dataset. You may need to modify the way the WHERE clause is built in order to do this.)

In addition to querying the list of filter values we also need to query the list of down drill selections (`DOWNDR_` instance variable) and across drill selections (`ACDR_` instance variable). These are both SCL lists and are the similar in content to the `SUBSET_BY_` list. The difference is that instead of containing sublists, these lists just contain named character items. For example a CALL PUTLIST on the `downdr_` list might yield:

```

DOWNDR_ ( &V11='DRUG_GROUP=Diabetic'
          &V12='DEPARTMENT=Central'
          ) [5311]

```

V11 and V12 are the metadata variable names. Again these will be formatted values in this list. Thus the process for creating a WHERE clause is similar to using the `SUBSET_BY_` list – walk through the list of variables, match up the variable with its metadata information returned from the `EMDDBMID_` instance variable, and build a WHERE clause to query the basetable using making use of any formats that may exist.

All the applicable WHERE clauses built in this manner from the filter values and down and across drill downs are inserted into an

SCL list. This WHERE_LIST list is then inserted as a named item in the EMDDbMid_ object (see above) making it available to any of the methods of our new MRV class.

Note that all of the above mentioned code to create the WHERE clause was done in the overridden _displayValues method of the MRV class. As it is rather lengthy the complete override of the _displayValues method that creates this WHERE_LIST is upon request from the authors.

APPLYING THE WHERE CLAUSE AND CREATING THE EXTRA TABLE.

Once the WHERE_LIST is built it can easily be applied to the source data set of the Mddb. The source dataset name is obtained by querying the metabase information:

```

/*****
/* Query the Metabase to get the Base Table */
*****/

alist=makelist();

call send(getnitemn(envlist('L'),
                        'METABASE',1,1,0),
          'GET_DATA_ATTR_',
          rc,
          msg,
          metabase_,
          mddb_,
          alist);

basetable = getnitemc(alist,'BASETABL',1,1,'');

```

Using the BASETABLE and the WHERE_LIST generated previously, one can create a WORK dataset, which we called WORK.TABLEDATA, that reflects the current subsetting state of the MRV. We created a new method for our MRV subclass called SUBSET_BASETABLE. In here, we used SCL submit blocks to generate PROC SQL code that creates a WORK dataset from the BASETABLE. We looped through the WHERE_LIST including each clause in the WHERE clause of the PROC SQL. Once the WORK dataset is available it can be used by any subsequent methods.

We applied an override to the _outputHtmlBefCloseBody method allowing us to add additional output to the HTML page after the MRV but before the closing </BODY> tag of the page.

We created a new method for the MRV class that we call from inside the overridden _outputHtmlBefCloseBody method. We first would use the ANALVARLIST (see above) to obtain the name and format of the current analysis variable in the MRV. This would allow us to construct PROC TABULATE code sending the results to the ODS destination. Here is the complete method used to add an extra table to the MRV report:

```

EXTRA_TABLE_1:

method;
put 'In Extra Table 1'; /* For Debugging */

css = symget ( "CSS" ); /* Cascading Style
                        Sheet Macro Variable */

/* The ANALVARLIST was created previously and
contains information about the analysis
variable*/

stats_list=getniteml(getniteml
                    (emddbmid_,
                    'TABLE_LIST'),'ANALVARLIST');

stats_item =getnitemc stats_list, "NAME" );
stats_format=getnitemc(stats_list, "FORMAT" );

```

```

/* Submit the PROC TABULATE code */

submit continue ;

ODS HTML body = _webout ( no_top_matter
                          no_bottom_matter )
                  stylesheet = ( URL = "&css" ) ;

options nocenter nodate nonumber ;

title '<H2>Sales Person Totals</H2>' ;

proc tabulate data = WORK.TABLEDATA
  formchar='82838485868788898a8b8c'x
  style = [ htmlclass = 'tdcell' ] ;

  table ( sales_person =
    [ label = ' '
      style = [ htmlclass='rowlab' ] ]

    all * [ style =
      [ htmlclass = 'tcolcell']] ,

    &stats_item = [ label = ' '
      style=[htmlclass='rowlab'] ] *
      f=&stats_format

    / box = [ label = " "
      style = [htmlclass = 'rowlab'
        nobreakspace=on] ] ;

  var      &stats_item ;

  class    sales_person /
    style=[nobreakspace=on] ;
  classlev sales_person /
    style = [ nobreakspace=on
      htmlclass='rowlab' ] ;
  keylabel all = "TOTAL"
    sum = ' ' ;
  keyword all /
    style=[htmlclass = 'trowlab' ] ;
  keyword sum /
    style=[htmlclass = 'rowlab' ] ;

run;

ODS HTML close ;

endsubmit;

endmethod;

```

There are a couple of items worth noting here. First of all, notice the HTML destination is open with NO_TOP_MATTER and NO_BOTTOM_MATTER. This is because the HTML output is to be dropped into the middle of an existing HTML page (the MRV report page). Also notice how the PROC TABULATE make use of the same styles classes the MRV uses, e.g. 'tdcell', 'rowlab', etc. This gives the extra table the same look as the MRV. Here is the extra table we created below the MRV:

The screenshot shows a web browser window displaying a SAS report. The main table has columns for various sales metrics. Below the main table, there is a section titled "Sales Person Totals" which lists individual salespersons and their corresponding totals.

Sales Person	Total
John D.	\$123,456
Jane E.	\$123,456
John F.	\$123,456
Jane G.	\$123,456
John H.	\$123,456
Jane I.	\$123,456

CREATING A PRINTABLE VERSION

Our reports were extremely wide, thus they were truncated when printed from the browser. Our clients needed to make hard copies of the complete report. Our strategy was to use PROC TABULATE to create tables using same layout as the MRV, send the output to an RTF destination allow the user to print it. The RTF destination would manage the pagination for us.

The most difficult task was to query the MRV class to get enough information about its current state in order to build a syntactically correct PROC TABULATE. We could get the information on the analysis variable using the same techniques used to draw the extra table (see above). What was missing was the list of class variables in the down and across dimensions.

We tackled each dimension separately. First we used the APPLIST attribute of the model class associated with the MRV, EMDDBMID_ (see above). This list contained information about the current state of the MRV. In particular it contained a sublist named "DOWN", which is a sublist of the "DIMENSIONS_L" list. As its name suggests it contains the list of down dimensions. For example it might look like this:

```
DOWN ( SALES VIEW=( HIERARCH='Sales View'
                    ) [19439]
```

It might just contain a single named item in which case there is only one class variable going down. In that case we could easily obtain the single class variable name. The difficulty occurred when the down dimension was a hierarchy as in the example above. In that case we needed to determine:

- 1) What is the current level of the hierarchy?
- 2) Are there any lower levels?

To answer #1 we first looked at the CURRENT_LEVEL item, a named item nested several lists deep inside the APPLIST. Once we determined the current level of the hierarchy we needed to answer #2. To this we looked at the EXPAND_ list attribute of the MRV class. This list contains all the values of the current level variable that are expanded. Here we make use of an assumption based on the current limitations of the MRV. Namely that the MRV can only display two class levels of a hierarchy at one time and only the first class variable can be expanded. Thus, if the length of the EXPAND_ list is greater than one then there must be another level. To determine the next level we queried the COLUMNS list, which is found nested several lists deep in the APPLIST. Using this information we created our own list of the class variables currently used in the down dimension. We use the same technique to determine the variables in the across dimension. From here we use SCL submit blocks to create the

PROC TABULATE using the subsetting MDDB base table as the source data.

Before the PROC TABULATE executes we turn on the RTF destination, sending the resulting output to file on the server. We used the SESSIONID (which is discussed later) to uniquely name the file. By overriding the _displayDefaultTitle of the MRV class we put out a hyper link to the RTF file so that it would open in new browser window or in a word processing document depending on the operating system. From here the user can easily print the report. A sample report is listed below:

The screenshot shows a web browser window displaying a SAS report. The main table has columns for various sales metrics. The report is titled "Single-Line Report" and "By Group Name".

Group Name	SALES	APPROX.	MAINT.	REPAIRS	TOTAL	AVG.	STDEV.	MAX.	MIN.
Group 1	\$14,155	\$14,155	30	80	90	90	80	90	80
Group 2	\$17,803	\$17,803	30	80	90	90	80	90	80
Group 3	\$15,884	\$15,884	30	80	90	90	80	90	80
Group 4	\$7,755	\$7,755	30	80	90	90	80	90	80
Group 5	\$19,800	\$19,800	30	80	90	90	80	90	80
Group 6	\$14,798	\$14,798	30	80	90	90	80	90	80
Group 7	\$20,183	\$20,183	30	80	90	90	80	90	80
Group 8	\$20,303	\$20,303	30	80	90	90	80	90	80
Group 9	\$12,379	\$12,379	30	80	90	90	80	90	80
Group 10	\$20,181	\$20,181	30	80	90	90	80	90	80
Group 11	\$1,341,12	\$1,341,12	30	80	90	90	80	90	80
Group 12	\$20,303	\$20,303	30	80	90	90	80	90	80
Group 13	\$6,775	\$6,775	30	80	90	90	80	90	80

As the code to determine the down and across land create this RTF report is rather lengthy, contact the authors to receive a copy of the complete source code.

KEEPING SESSION VARIABLES

As mentioned above we needed to use the _SESSIONID macro variable. Our application ran from inside another SAS/IntrNet application that used pool services with sessions. This was done for security and other application reasons. In order for our application to stay connected to the sessions we need to make sure there were three parameters sent to the broker with every request (i.e. with every drill down or filter value change). These parameters were _SESSIONID, _PORT, _SERVER. We needed to modify three methods in order to write these parameters out to the HTML coming back from the server so that they would be available to send back to the server with the next request. The methods we needed to override were: _outputUriOptions, _outputHiddenFields, and _outputAddtClvalParms. In each of these methods the override is very similar. We first perform a CALL SUPER to allow the method to execute its default code. Then we add additional to output these extra columns. In the case of the _outputHiddenFields methods they are added as hidden HTML fields, for _outputAddtClvalParms they are output as JavaScript variables and for the _outputUriOptions they are output as name/value pairs.

CONCLUSIONS

The SAS Multidimensional Report Viewer is an easy out of the box solution to surface reports on the web. However, we found that we needed to make several major overrides to the behavior and appearance of the MRV in order to implement it in a high demand production environment. Many of these changes were very straightforward. Others were much more complex requiring significant investigation to discover the content of instance variables and behavior of methods that were not always well

documented.

REFERENCES

Harris, Colin (2001), "Dynamic Ad-Hoc Reporting via the Web: Real World Comparison of the MDDB Report Viewer and AppDev Studio®", Proceedings of the Twenty-sixth Annual SAS Users Group International Conference, Paper 184-26.

Katz, Mary Frederico and Rood, Bob (2002), "Web-enabling a Client/Server OLAP Application Using SAS/INTERNET® Software's MDDB Report Viewer", Proceedings of the Twenty-seventh Annual SAS Users Group International Conference, Paper 28-27.

SAS Institute Inc. Web Site URLs for online documentation:

MDDB Report Viewer:

<http://www.sas.com/rnd/web/internet/mddbapp/index.html>

MDDB Report Viewer Macro Variable and Global Variables:

<http://www.sas.com/rnd/web/internet/mddbapp/webvars.html>

MDDB Report Viewer Class, Instance Variables, Flow of Control, and Methods:

<http://www.sas.com/rnd/web/internet/mddbapp/methods/index.html>

SAS/IntrNet Documentation Index:

<http://www.sas.com/rnd/web/internet/sitemap.html>

ACKNOWLEDGMENTS

We gratefully acknowledge the technical support staff at the SAS Institute for their guidance as we worked numerous issues.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Also all of the source code needed to perform the customizations mentioned in the paper is available upon request from the authors. Contact the author(s) at:

Alan Leach
Qualex Consulting Services, Inc.
2217 Cleeneay Ave
Norwood, OH 45212
Work: 513.731.8831
Fax: 603.807.5671
Email: Alan.Leach@qlx.com
Web: www.qlx.com

Steve Wagar
Qualex Consulting Services, Inc.
659 Lake Hills Dr.
Trenton GA 30752
Work: 706.657.2377
Fax: 603-658-0653
Email: Steve.Wagar@qlx.com
Web: www.qlx.com