Paper 41- 26

A SAS® Based Correspondence Management System

Bernd E. Imken, Patented Medicine Prices Review Board, Ottawa, Canada

Figure 1 - The Original Version 6 DataForm Application - BEFORE MODIFICATIONS DISCUSSED IN THIS PAPER

ABSTRACT

The tracking and management of correspondence, especially customer complaints, has become a high priority for most organizations. The recent focus on Customer Relationship Management has augmented this requirement. The application discussed in this paper was developed using SAS/AF. The application resides on a Windows NT based server and uses SAS/Share with TCP communicationsl. The paper discusses the ways in which a correspondence management system may be developed using SAS/AF Version 8 software.

INTRODUCTION

The Patented Medicine Prices Review Board (PMPRB) was established as a judicial agency by the Canadian Government to monitor the prices of patented medicines sold in Canada and to review the level of research and development expenditures by pharmaceutical companies. The PMPRB currently monitors the prices on nearly $6 Billion sales of patented medicines in Canada

each year. The mission of the PMPRB is to contribute to Canadian health care by ensuring that prices of patented medicines are not excessive.

Tracking incoming correspondence has been done at the PMPRB for many years. Earlier, the PMPRB had used a computer system developed by a external software vendor. When this system proved not to be compliant with Year 2000 standards, a decision was made to develop a customized in-house application which would meet the Board's requirements.

Requirements included:
!   track all correspondence
        "   incoming
        "   outgoing
!   classification
        "   by client/author demographics
        "   by media
        S   mail, FAX, telephone etc.
        "   market segmentation
        S   public, association, manufacturer,

government sector etc.
"   type of correspondence
     S   inquiry, complaint, data
     submission, etc.
"   language
     S   English, French
"   other...
!   tracking capabilities
"   automatic Bring Forward dates
"   automatic Due dates
"   other dates
     S   Completion, date on document,
     response date, review dates etc.
!   access must be available simultaneously for all employees
!   flexible reporting
!   etc.

The first version of the Correspondence Management System was developed using SAS Version 6.12. This application develped a SAS/AF multipage DataForm object. Development of such an application was discussed in the Sugi 24 paper#239 "The Use of DATA FORMS, TABLES and other new OBJECTS in SAS/AF". This paper will not discuss how the 6.12 version was developed because SAS now has new approaches to building such applications. In general, data forms have been replaced by Form Viewer Controls and data tables have been replaced by Table Viewer Controls How these objects are treated using Version 8 are completely. This paper will instead demonstrate how a system such as the above could be developed in simple steps using the improved facilities for Version 8.

BUILDING THE VERSION 8 APPLICATION

The application discussed below will be a simplified version of the screen displayed above. It will not show the reader all the variables on the screen, but will instead focus on how the basic system can be engineered. It is recommended that any user attempting to build such as system first learn some of the basics of SAS/AF development, including use of the BUILD procedure, selecting COMPONENTS, and writing basic SCL code. These basic concepts can easily be mastered by taking an introductory SAS/AF course. Users can also be quickly introduced to these concepts by attending the Sugi Presentation on this paper.

The Basic Steps
!   **STEP #1**
"   Using a Data Step to create the initial Master File
     S   creating the basic variables
     S   indexing and compression
"   Applying Proc DataSets
     S   creating integrity constraints
     S   creating an audit trail
!   **STEP #2**
"   Using Proc Build to create the Frame
     S   creating the form viewer
     S   adding the SAS data set model
     S   modifying properties
     #   identifying the data table
     #   changing the design
     S   creating a combo object
     S   other design changes
     S   testing the application so far
!   **STEP #3**
"   Using a Data Step to create a transaction file
     S   creating the basic variables

!   **STEP #4**
"   Linking the Master file to the Transaction file
     S   METHOD 1 - The transaction file is created within the Form Viewer object
     #   as another Form View
     #   as a Table View
     S   METHOD 2 - the transaction file is created as a separate object outside of the Form Viewer
     #   two approaches shown
!   **STEP #5**
"   Migrating the application from single user mode to multi user
     S   Installation of a SAS Server and changes to Client.
!   **STEP #6**
"   Using ODS output to generate high quality listings
     S   using ODS PRINTER
     #   a sample application report
     #   generating an audit report

STEP #1
In order to create this simplified application a SAS data set must first be created to store the input data. The following SAS code is used to created the MASTER data set:

```
data sasuser.master(index=(mailno / unique)
              compress='yes');
    mailno='    ';
    name='               ';
    address='                    ';
    type='         ';
    class='            ';
    label mailno="Mail Identification #"
        name="Author's Name"
        address="Address"
        type="Correspondence Type"
        class="Classification";
run;
```

This example data set has 5 variables, initially 1 observation is created. Two data set options have been specified. The first being to index the dataset using the variable mailno, the second to compress the data set. Compression was used successfully in the production version of the PMPRB Correspondence system to eliminate blanks etc. - savings were achieved greater than 50% without significant performance degradation. (Note: indexing and compression options can also be set using PROC DATASETS)

Next, let's add some data to the data set that we just created. Issue the following command on the command line.

```
VIEWTABLE sasuser.master
```

Remember to click on EDIT on the menu and select "row edit" and "ADD a row". After you have entered a number of rows the screen should look like the one in figure # 2

Figure 2 - adding data with Viewtable

Next we will issue the following SAS code to create integrity constraints;

```
proc datasets lib=sasuser;
    modify master;
    integrity constraint
        create check_key=PRIMARY KEY(mailno)
                message='You must provide a
                    UNIQUE identifier';
    integrity constraint
        create check_type=CHECK
                (where=(type in ('Letter'
                            'FAX')));
    run;
```

The usage of integrity constraints in Version 8 significantly improves error checking - reducing the amount of programming required. In the example above, the data input for mailno is checked to insure that it is unique, otherwise a message is generated for the user. This check is applied whenever The variable type is also checked to make sure it is one of the two permissible values.

Next, we will again use PROC DATASETS to create an audit trail for the master file.

```
proc datasets lib=sasuser;
    audit master;
    initiate;
    run;
```

Now that the master file has been created we are ready to create the form viewer within SAS/AF.

STEP #2
To access SAS/AF issue the following command in the program editor:

```
proc build c=sasuser.sugi26;run;
```

This will create a new AF catalogue if one does not already exist by this name. Once the catalogue is created click on the menu and open up New under the File command. Next select the Frame icon. The system will open up a new frame for you. Proceed to the COMPONENT window and select FORM
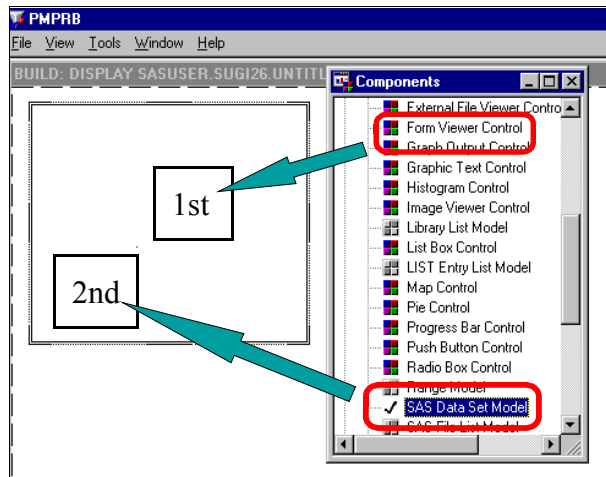

Figure 3 - Creation of the Form Viewer with the SAS Data Model attached

VIEWER CONTROL for the list; drag it onto the newly created frame. An empty rectangle will be generated which may be resized for the size of the screen which you desire.

1st	Click the Form Viewer Control located in the Components windows and drag it onto the frame. This will create the larger rectangle which is displayed in Figure#3. This rectangle can now be resize to your requirements on the screen.

2nd	Click the SAS Data Model located in the Components windows and drag it to the frame dropping it over the retangle created above. You will not notice and change at this point in the appearance of the rectangle.

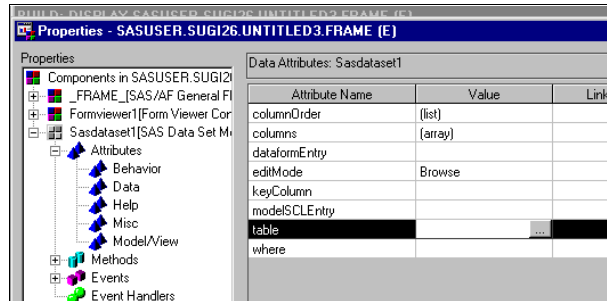| 3rd	Click your right mouse button in the rectangle generated when you creater you Form View and select "PROPERTIES". SAS will open the Properties window. Drill down on the properties for sasdatset1, then on Attributes, and then finally click on Data to display the screen shown in Figure #4.

Figure 4 - The Properties Window

Next proceed to the attribute called TABLE and type in SASUSER.MASTER or clcik on the icon the make the selection. Once this has been done the properties window changes to display all the new fields which are in the data set, together with objects for their labels. This is demonstrated in Figure#5.
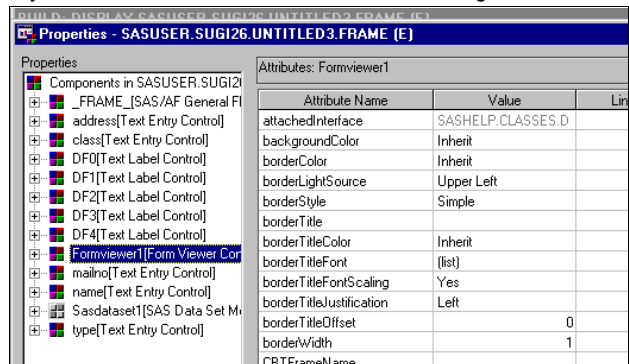

Figure 5 - The new fields have been added to the Form Viewer

Once you exit the Properties window you will discover that the Form Viewer now has been updated with all the variables and associated labels in the data set which you created earlier.
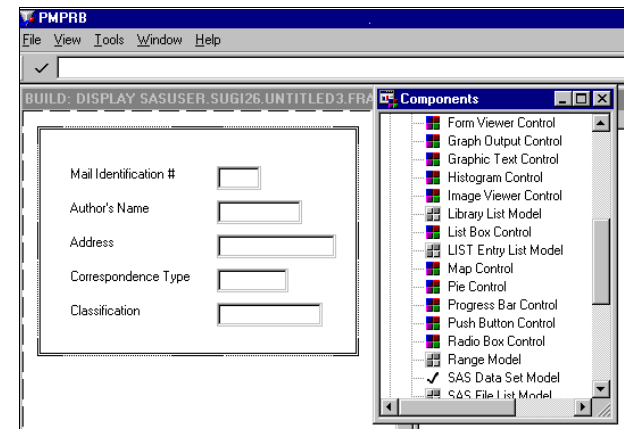

Figure 6 - The updated Form Viewer

4th

Next, click on Build on the main menu and select TESTAF to test the form viewer that you have just created.
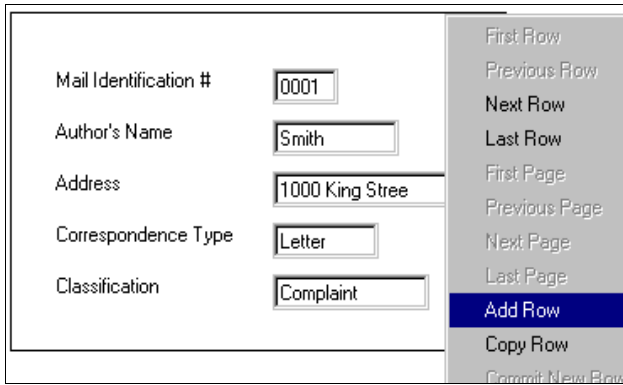

Figure 7 - TESTAF with Menu

When the right mouse button is clicked the system displays the menu as shown in Figure #7.

Clicking on EDIT allows you to ADD new records to the MASTER data set.

Let's next change the screen so that type will have a pull down instead of a text box. First click on the existing text box, press the right mouse button and delete it. Then proceed to the Component window and select the COMBO BOX CONTROL and drag it onto the form viewer. Right click the combo box which is created and again select PROPERTIES.

Drill down to DATA for COMBOBOX1 and then click the icon for the attribute ""items". Figure #8 will now be displayed, allowing

you to add the selection values to the combo box. It is also useful to change the name property from the default COMBOBOX1 back to "type". There is however one addition thing to do before it works, it will still be necessary to associate the newly created combo box
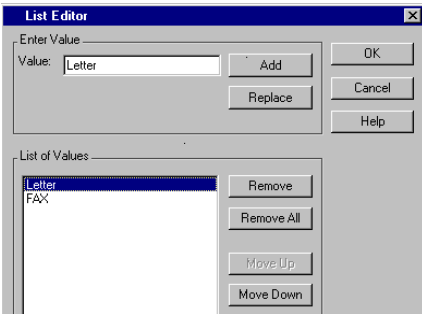
Figure 8 - Generating a Combo box

with the "type" variable in the data set. This is accomplished by right clicking the form viewer, selecting "Form->" and then selecting "Display Column Window", the screen show in Figure #9 will be displayed.

Within this screen, click on type and then drag it over the newly created combo box. Now, if you test the screen the value for "type will appear appropriately in the combo box.
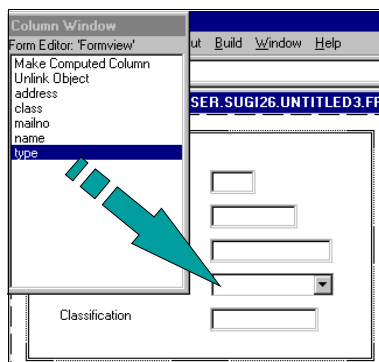

Figure 9 - Associating a data set variable with a form viewer object

This method of associating variables must also be done if new variables are subsequently added to the form viewer or other modifications are made.

Figure 10 - Testing the combo box

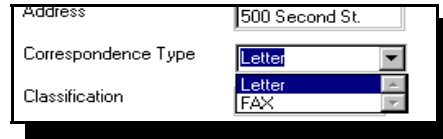Another useful method of populating a combo box is to drag over a "**Variable Values List Model**" from the component window over a newly created combo box. Using this method the box become populated with the values from a SAS data set. This is much more convenient for larger lists, it also makes maintenance easier. Similarly to the approach used with the **"SAS Data Set Model**" it is also necessary to change the value of the "table" to point to the SAS data set.

STEP #3
Let's next create a transaction file, populate it and link it to the master file formviewer which we have created.

Figure 11 -The transaction data set

```
data sasuser.distribution
    (index=(mailno));
      mailno='     ';
      employee='          ';
   run;
```

The transaction data set was popluted using viewtable.

For this example, neither compression nor an audit trail was developed. In a production environment both would probably be done.

STEP #4
Method #1 - Creating a form or table viewer _within_ a form viewer.
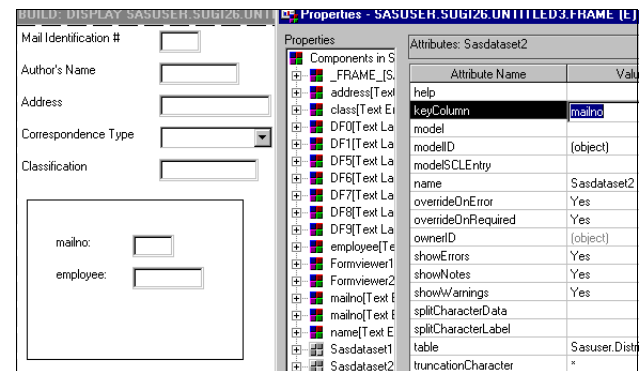

Figure 12 - Creating a second formviewer within the first

Let's now create another form viewer within the one which we had originally done. This is demonstrated in figure #12. It is created the same way as we did in Step#2. Note that the table attribute for sasdataset1 has been set to sasuser.distribution and the keyColumn attribute has been set to mailno. One final instruction remains. To link the second formviewer with the first it is necessary click the first formviwer created to make it active and then to issue the following command:

```
link formviewer2 mailno
```

This must be typed on the command line. It is also possible to link a table viewer to the original formviewer. This is even easier. It is only necessary to drag the "`mailno`" variable from the
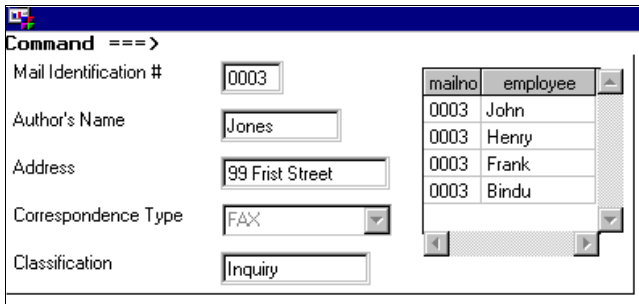

Figure 13 - Link with a Table Viewer

"Column Display Window" associated with the original fornviewer over the "`mailno`" column in the tableviewer to establish the linkage.

Method #2 - Creating a form or table viewer externally on the same frame. To do this it is necessary to write some SCL code. Two approaches can be used.

Figure 14 - 2 Separate Linked Form Viewers

Both approaches are shown since there are instances when it is more convenient to use one rather than the other. The result of either approach is the same and is demonstrated in Figure #14.

Approach 1: The code could be written solely at the formviewer level. No special programming is required at the frame level. First the "`ModelSCL`" attribute must be changed on the formviewer to point to SCL for the viewer. The code for the viewer is show below:

```
DFINIT:
   dcl object formviewer2 modelid;
   frameid=getnitemn(_viewer_,'_frame_');
   call send(frameid,'_getWidget',
         'formviewer2',formviewer2);
   modelid=formviewer2.modelid;
     vals=makelist();
     return;
   INIT:
     link mailno;
     return;
   MAILNO:
     rc=clearlist(vals);
     rc=insertc(vals,mailno,-1,'MAILNO');
     rc=modelid._setKey('mailno',
            'EQ','SCROLL',vals);
     return;
   DFTERM:
     vals=dellist(vals);
     return;
```

This approach identifies the second fornviewer to the first in the DFINIT section and then uses the "_setKey" method to link the second formviewer to the first each time the "mialno" changes.

Approach 2:
This approach requires the follow code at the frame level.
```
   INIT:
     control always;
```

```
   formviewer1._setEventHandler
      (formviewer1,'FV_ROW_CHANGED',
      '_set_key_',formviewer2);
   sasdataset1._rereadCurrentRow();
   return;
```
Then at the formviewer level the following ModelSCL is required:
```
   DFINIT:
     vals=makelist();
     return;
   INIT:
     link mailno;
     return;
   MAILNO:
     rc=clearlist(vals);
     rc=insertc(vals,mailno,-1,'MAILNO');
     _viewer_._sendEvent('FV_ROW_CHANGED',
        'MAILNO','EQ','SCROLL',vals);
     return;
   DFTERM:
     vals=dellist(vals);
     return;
```
With the technical capabilities demonstrated above it is possible to develop a wide range of applications. The PMPRB developed an enhanced version of its Correspondence version using the techniques shown. One major thing however is still missing for such an application to be utilized effectively - the ability of multiple users to access a SAS data set simultaneously. This is accomplished by the use of SAS/Share software which is discussed in the next portion of this paper.

STEP #5
The PMPRB installed a separate copy of SAS for an NT Server (Note: this requires a separate licence/installation specific for an NT server). The PMPRB has this server connected to a LAN with TCP/IP. SAS running on this server allows multiple SAS users to be connected to thi server to share one or many SAS data sets or catalogs simultaneously.

The SAS data sets and catalogs which were developed were copied to the SAS Server and were stored on one of the drives on this server. The SAS/NT server was called "sas_ser2" . When SAS is started up on this server a procedure needs to be excuted as shown below:
```
   Proc server serverid=sas_ser2
   authenticate=opt;run;
```
Libraries which are allocated with LIBNAME statements to this server now become available to others connected to the LAN in SHARE/muti-user mode.

In order to gain access to these shared data sets the LIBNAME statements on the client (usually stored in the autoexec.sas file) must be altered as shown in the statement below:
```
      option comamid=TCP set=vqpname &userid;
      libname pmprb 'd:\dbshare'
         server=sas_srv2.sas_ser2;
```
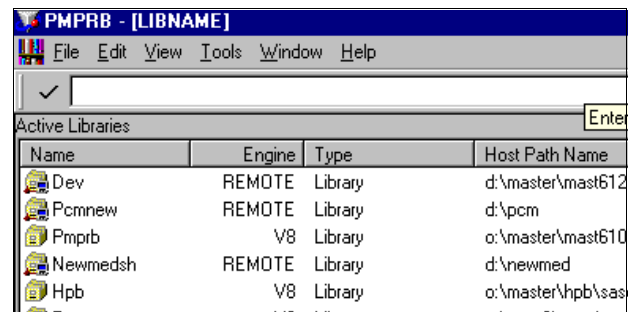

Figure 15 - SAS/Share libraries

Where, pmprb is the newly allocated libname for a folder on the NT server. The folder is located on the "d:" drive of the server. The NT server is called sas_srv2 and the running SAS server

session is called sas_ser2. If users were to do a LIB command they would see any libraries on the SAS/Share server identified as being on a REMOTE engine.

If you examine the NT server running you will see each access to the server is logged as is shown in a running session on the PMPRB NT/Share server as shown in Figure #16. When a data set is running in SHARE mode and a user accesses a SAS observation while the SAS data set is locked at the record level,
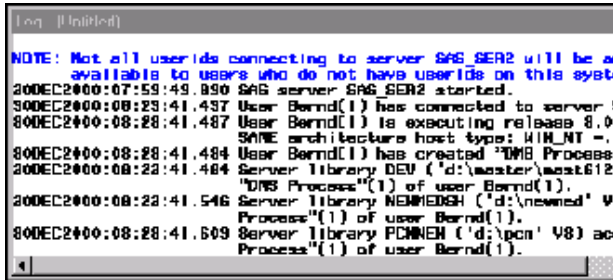

Figure 16 - Log from a running SAS/SHARE server

he or she will gain exclusive access to change that record. Any other user accessing that record at exactly the same time will receive a message that the first user currently has that observation under his or her control. Similarly users can access that same data set simultaneously for reporting purposes, any records which are currently being locked will use the before-image of that observation for reporting purposes.

STEP #6
Now let's generate the first report for the applications which we just developed. The Output Delivery System will be demonstrated since it was used to developed quality reports for the new PMPRB application. Sample reports are provided using the application which we have developed for this paper.
If HTML output is switched on and the following SAS code is executed:

```
PROC REPORT DATA=SASUSER.MASTER LS=163
  PS=50  SPLIT="/" NOCENTER ;
   COLUMN  mailno name address type class;
   DEFINE  mailno / DISPLAY FORMAT= $4.
     WIDTH=14 SPACING=2 LEFT
          "Mail Identification #" ;
   DEFINE  name / DISPLAY FORMAT= $10.
     WIDTH=10 SPACING=2 LEFT "Author's Name";
   DEFINE  address / DISPLAY FORMAT= $15.
     WIDTH=15 SPACING=2 LEFT "Address" ;
   DEFINE type / DISPLAY FORMAT= $8. WIDTH=14
     SPACING=2 LEFT "Correspondence Type" ;
   DEFINE  class / DISPLAY FORMAT= $13.
     WIDTH=14 SPACING=2 LEFT
          "Classification"; ;
   RUN;
```
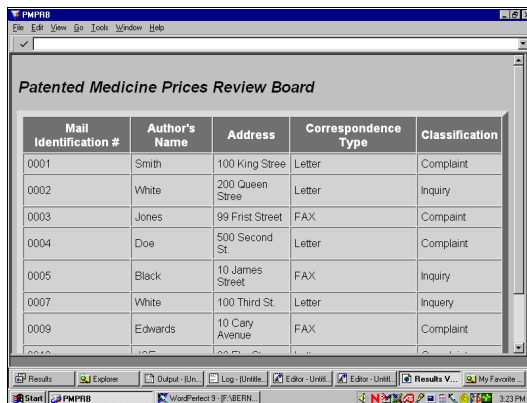

Figure 17 - ODS Report Output   using D3D Style

Using ODS it is possible to further improve the appearance of the report be modifying font, spacing etc directly within the code of Proc Report.

```
style(column)={background=grayF0 cellwidth=120
               font_weight=BOLD};
```
The sascode above could be added to the DEFINE statement to change the appearance of a column. It is also possible to send reports directly to your printer by specifying and executing the following SAS code prior to the report:

```
              ODS PRINTER;
```
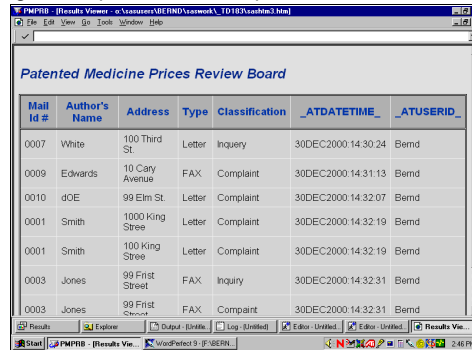Below you will see another report done using a different style showing the output for a audit report:

Figure 18 - Example of Audit Report using the Default style template

This report displays the audit time stamp and the audit trail for the userid which made a change - there are also many other variables which could be reported - including before and after images of records, whether the change was an ADD or a modify etc..

CONCLUSION
The paper demonstrates how SAS can be used to build a Correspondence Management Application. This application that is in production has now been running for over a year. This paper shows how you too can easily develop a similar model. The application used at the PMPRB also includes PMENU support etc, and is more complicated than can be presently within the confines of this paper.

The techniques discussed in this paper can also be easily adapted to many other applications. Other applications which have been developed at the PMPRB in a similar manner include: a hot line / problem tracking system, used to monitor hardware and software problems as well as, a system to track periodicals.

ACKNOWLEDGMENTS
I would like to thank Ann Rockett from SAS Institute for her help. Assistance, together with examples, that she has provided me has been invaluable.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

Author: **Bernd E. Imken**
                 Chief, Information Systems Division

Address: **Patented Medicine Prices Review Board**
                 333 Laurier Avenue West,
                 Ottawa, Ontario, Canada.   K1P 1C1

Work Phone: (613) 952-3312
Fax: (613) 952-7626
Email: imken@sympatico.ca
Web: www.pmprb-cepmb.gc.ca