Error Handling: An Approach for a Robust Production Environment
Burak Sezen, Health Dialog, Portland, ME
Sandeep Gaudana, Health Dialog, Portland, ME

ABSTRACT
Error handling is one of the most important ingredients of a good program. Timely detection and handling of error can save time and resources and prevent potentially many other problems down the road. This paper focuses on detection of errors, determining steps to be taken based on its severity and finally communicating the occurrence of error to the concerned people. SAS e-mail feature is used as a means of communication. Macros used by us as part of the Error Handling Framework are included.

INTRODUCTION
Health Dialog Analytic Solutions (HDAS) uses SAS software to extract data being provided in many different formats by a variety of clients, to cleanse and eventually to transform them to be loaded into standard table structures. These tasks require complex programs requiring and consuming extensive resources both in terms of processing power and storage. In this type of environments, it is imperative to have highly optimized and well controlled processes.

To get an idea of impact that an error can cause to a production job consider a basic example:

```sas
proc sort data = liba.members_2005 out = members_2005;
   by member_id;
run;

proc sort data = liba.members_2004 out = members_2004;
   by member_id;
run;

Data output;
   merge members_2004(in=in1) members_2005(in=in2);
   by member_id;
   if in1 and in2;
run;
```

In the above program, an “output” dataset is created by merging “Members_2005” and “Members_2004” datasets. But due to some reason “liba.Members_2005” does not exist and an error is generated. However, the process does not stop at that point and SAS moves forward to sort “liba.Members_2004” dataset. That in turn generates another error when the “Members_2004” and “Members_2005” datasets are attempted to be merged. Ultimately, the goal of creating an “output” dataset is not achieved and the processor time to sort “liba.Members_2004” dataset is wasted. The impact can be greater when trying to run production jobs requiring significant resources dealing with larger datasets.

This paper discusses a mechanism HDAS uses to handle run-time errors that might occur during the execution of batch SAS processes. It does not provide any functionality for handling errors in an interactive development environment.
HOW ARE ERRORS TRADITIONALLY HANDLED IS SAS?

Error handling is one of the most important ingredients of a good program. Timely detection of error and proper handling can save time and resources and prevent potentially many other problems down the road.

There are five types of errors in SAS:

1) Syntax
2) Semantic
3) Execution Time
4) Data
5) Macro Related

The first two types of errors are detected at compile time. SAS won’t run if there are compile time errors. The last three types of errors are runtime or execution time errors. See SAS Documentation, cited in the Reference Section for further detail on the types of errors.

The runtime errors are conditional and occur due to an unexpected complication during the execution of the process. Many contemporary programming languages provide the necessary mechanisms that respond to this type of errors. These mechanisms can perform multiple tasks such as informing the user of the erroneous or exceptional situation, providing alternatives for dealing with it, and branching out based on the type and/or severity of the problem or allowing the program to continue its normal operation and prevent it from crashing and displaying of cryptic error messages to the user. One other advantage of proper error handling is that it can point the developer to the exact problem and eventually save time and resources necessary for debugging and trouble shooting efforts.

SAS has introduced contemporary error handling capabilities in SCL with Version 8 (see David Scocca’s SUGI 29 paper, cited in the References Section for further detail on this capability of SCL). However, there still isn’t a good way in the base product to catch runtime errors, and handle them properly resulting in a graceful termination of the process. SAS provides capabilities to stop a process in case of an error either at the data step or at the session level. The former is achieved through the ABORT ABEND/RETURN statement used within the data step. ABEND causes abnormal termination of the current SAS job or session, while RETURN causes the immediate normal termination of the current SAS job or session. The latter is achieved by setting the ERRORABEND system option. ERRORABEND causes an abnormal termination of the SAS job running in case of an error for an interactive session. It goes into syntax check mode in batch mode.

Even though, these are very valid ways for handling errors, they have the following drawbacks:

1) In case of jobs running in production mode with minimal monitoring, this approach does not provide any clues that an error has occurred during the execution.

2) It is up to the operator to review the saved log files to locate the error messages in order to understand the cause of the problem. This might not be even possible if the log file was not saved.

3) In case of ERRORABEND system option, it does not allow the process to perform certain tasks before the termination of the SAS session.

This leaves the users with no tools to control errors within a process and decide to stop or continue programmatically. Over the years different SAS shops have come up with creative solutions for this problem:

1) COMMENTING CODE DYNAMICALLY

The first approach is to introduce a mechanism, which will void the remainder of a program when a runtime error occurs. A macro, which starts a dummy macro definition is called within the program body. In case of an error, the program starts the macro definition, which makes the remainder of the code part of the macro. At the bottom
of the program a “%mend” statement is executed, which completes the macro definition. Since the newly defined dummy macro is never called, the code it contains is never executed, which ends up being the desired outcome. Even though, this is a smart approach, it has a couple of disadvantages. It does not inform the operator about the error and the cancellation of the process. It also causes the color coding in the SAS editor disappear. For further details, see Christian Graffeuille’s Quick Tip for Cancelling Processes in the References Section.

2) USING A WRAPPER SAS JOB

Another interesting approach is to have a SAS “wrapper” process calling other SAS programs as separate SAS sessions using the SAS SYSTASK statement. This way even when one of the programs errors and therefore terminates, the “wrapper” SAS session continues and can perform post error tasks or even continue from the point the process failed. The disadvantage of this approach is that the processes need to be broken into separate SAS programs, which might not be always desirable. See Denis Cogwell’s SUGI 30 paper cited in the References Section for further detail on this approach.

HDAS SAS PROCESSING FRAMEWORK

HDAS has developed a set of macros to manage all aspects of the production cycle. A process is broken into three sections:

- Header
- Body
- Footer

The header section is used to setup the environment for the process to be executed. This involves establishment of any remote connections, inclusion of macro libraries and initialization of macro variables referred by the process during the execution.

The body section has the main code to be executed. All the process related data steps, procedures and calls to other macros are included in this section. Any type of error checking and code branching due to process problems are also handled in this section.

The footer section is where the post process tasks are performed. These tasks include completion of output and log streaming, any type of library cleansing, release of any occupied resources, collection of performance statistics and potentially messaging to the operator informing the completion of the process.

HDAS ERROR HANDLING

A SAS process has many similarities to an auto manufacturing assembly line. A car being manufactured undergoes a set of processes in a specific order. If due to some problem one of these processes fails then the entire assembly line stops. This might cause serious losses in production and time but it also prevents the production of a car with faults. HDAS had contemplated these issues and had come up with the following approach:

The error handling process is broken into three separate macros, which work synchronously. Each of these three macros is called in a different section of the HDAS Framework discussed previously. Figure 1 shows the framework used for Error Handling.

THE INIT MACRO

The %init macro is called in the header section. This macro initializes a set of global macro variables, which will be referenced by the downstream %chkerr and %final macros. The parameters this macro takes dictate certain process behavior such as whether the SAS log and output content would be saved into external files, whether the operator will receive an e-mail in case of an error. The code for this macro is provided in Appendix A.
THE CHKERR MACRO

The %chkErr macro is used in the body section of a process. This macro performs the following tasks:

1. Checks the value of the syserr macro variable.

2. In case of an error, it skips the remainder of the process by calling the %final macro, which will be discussed later in this paper.

3. Otherwise it returns back to the main process from the point it was called.

It is up to the programmer's discretion when and how often to use this macro. It is typically called after the data steps or procedures, which are most prone to runtime errors and most costly in terms of time and resources in case of an error. If needed, this macro can be called after each data step and procedure in the SAS program. This is recommended considering that this macro is very light in terms of resource consumption. This macro has a parameter called checkpoint, which is used to pass a descriptive text to be included in the message sent to the user in an error condition. This is a good and efficient way of communicating the exact location where the process encountered an error. The code for this macro is provided in Appendix B.
THE FINAL MACRO

The %final macro, which is called in the footer section, performs the necessary steps to complete the process. This will involve closing the log and output being saved into external files, sending an e-mail to the user informing an error or completion status of the process that has been executed.

Based on the settings in the %init macro and errors occurring during the processing, the final macro follows the logic demonstrated below to determine the type of actions it should take:

As shown in the above flow chart (Figure 2), the operator receives an e-mail informing the status of the process. The message content is success in case of no errors. In case of an error, there might be two possibilities. If there is a
%chkErr macro call right after the data step or procedure causing the error, an e-mail with the error and “check point” message will be sent to the operator and the process will be terminated right after. However, if the %chkErr macro is not called after the point of error and the process completes with errors, the operator will receive an e-mail just with the error messages. This gives quite a bit of flexibility to how and when errors should be handled and whether the process should continue or not. The code for this macro is provided in Appendix C.

SAMPLE USAGE OF THE ERROR HANDLING MACROS

The situation cited by an example at the beginning of this paper can be handled by the Error Handling Macros in the following way:

```sas
%init(logfl = c:\nesug_2005\sample_log.txt, 
     printfl = c:\nesug_2005\sample_print.txt, 
     email = I, 
     Processname = TestRun);

proc sort data = liba.members_2005
   out = members_2005;
   by member_id;
run;

%chkErr(checkpoint = Members_2005);

proc sort data = liba.members_2004
   out = members_2004;
   by member_id;
run;

%chkErr(checkpoint = Members_2004);

Data output;
merge members_2004(in=in1)
   members_2005(in=in2);
   by member_id;

if in1 and in2;
run;

%final();
```

As shown in the above example, a destination for log file and print file is provided in the %Init Macro. The parameter email = I will allow an email to be sent. It has to be noted that only the user submitting the program will receive an email. As the dataset “liba.members_2005” does not exist, SAS will give an error. The %chkErr included after the sort procedure will detect this error and will stop SAS from processing any more steps. As the log file is saved at a destination, the user will get an E-Mail with the error message in the body of the E-Mail. %Final should be used at the end of the program. Thus, the Header Section Macro (%Init), Body Section Macro (%chkErr) and the Footer Section Macro (%Final) forms an Error Handling Framework for a Robust Production Environment.

CONCLUSIONS

The lack of a mechanism to handle run-time SAS errors is a problem shared across many SAS shops. HDAS has developed a mechanism to address this problem for production processes, which are kicked off as batches either remotely or directly on the servers. This paper does not provide any functionality for handling errors in an interactive development environment. Even though the macro codes were developed for a specific purpose, the approach is simple and generic enough to be implemented in other environments as well.
REFERENCES

SAS Documentation: SAS 9.1.3 Language Reference: Concepts


Graffeullie, Christian, “Quick Tip: Smoothly Canceling Processes at Will” Available at http://support.sas.com/sassamples/quicktips/03dec/cancelprocess.html


ACKNOWLEDGMENTS

SAS® is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina.

The authors wish to express their thanks to Judy Loren for her valuable suggestions on this paper.

DISCLAIMER

All code provided in this paper is provided on an "AS IS" basis, without warranty. Neither the authors nor Health Dialog make any representation, or warranty, either express or implied, with respect to the programs, their quality, accuracy, or fitness for a specific purpose. Therefore, neither the authors nor Health Dialog shall have any liability to you or any other person or entity with respect to any liability, loss or damage caused or alleged to have been caused directly or indirectly by the programs provided in this paper. This includes, but is not limited to, interruption of service, loss of data, loss of profits, or consequential damages from the use of these programs.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Burak Sezen or Sandeep Gaudana
Health Dialog
39 Forest Avenue
Portland, ME  04101
BSezen@HealthDialog.com or SGaudana@HealthDialog.com

*****************************************************************************
APPENDIX A – INIT MACRO

/**************************
* The %Init(), falls under the Header section of the Error Handling Framework
* 
* Macro Parameters:
* 
* @ logfl                 : Destination of Log File
* 
* @ printfl               : Destination for Output List File
* 
* @ email(0 or 1)         : Value of 1 sends an email at the end of the process
*   whose Performance Characteristics will be measured
* 
* @ Processname           : Description of the process that will eventually go
*   into the subject line of email
* 
* 
* Macros Called : None
* 
/**************************;

%macro init
  logfl=,
  printfl=,
  email=0,
  processname=);

options NOERRORABEND;

%global hdds_err_flg;       /* Error flag, set to 1 if an error occurs */
%global hdds_email_flg;     /* Send e-mail flag, if 1, an e-mail is sent to the operator in case of an error */
%global hdds_process_name;  /* The unique name for the process being executed */
%global hdds_log_file;      /* The path and name for the file log content is written to */
%global hdds_output_file;   /* The path and name for the file output content is written to */
%global error_log;          /* The number of Error Lines extracted from the Log File*/
/*Initializing the Macro Variables*/

%let hdds_err_flg=0;
%let hdds_email_flg=0;
%let hdds_log_file = ;
%let hdds_output_file = ;
%let error_log = 0;
%let hdds_process_name=%upcase(&processname);

%if (%eval(&email>0)) %then %let hdds_email_flg=1;

/* Start writing into external files if log or output file paths and name are
 provided */

%if (%eval("&logfl" ne "" or "&printfl" ne "")) %then %do;

proc printto
   %if ("&logfl" ne "") %then %do;
       %let hdds_log_file = &logfl;
       log="&logfl"
   %end;

%if ("&printfl" ne "") %then %do;
       %let hdds_output_file = &printfl;
       print="&printfl"
   %end;
   new ;
run;
%end;

%mend init;
APPENDIX B – CHKERR MACRO

/**********************************************************************************/
*                                                                                *
* The %Chkerr(), falls under the Body section of the Error Handling Framework.    *
*                                                                                *
* This macro can be used after any data step or procedure                         *
*                                                                                *
* Macro Parameters:                                                               *
*                                                                                *
* @ checkpoint            : Description provided by user to easily detect the     *
* data step or procedure that encountered error                                 *
*                                                                                *
* Macros Called : %Final                                                          *
*                                                                                *
/***********************************************************************************/

%macro chkErr(checkpoint=);

/*------------------------------------------------------------------------------
In case of an error with syserr greater than 0 but not equal to 4 then call the
%final macro and terminate

Syserr Description

 0   Execution completed successfully and without warning messages.
 1   Execution was canceled by a user with a RUN CANCEL statement.
 2   Execution was canceled by a user with an ATTN or BREAK command.
 3   An error in a program run in batch or non-interactive mode caused
      SAS to enter syntax-check mode.
 4   Execution completed successfully but with warning messages.
>4   An error occurred. The value returned is procedure-dependent.
-------------------------------------------------------------------------------- */

%if (%eval(&hdds_err_flg=0 and (&syserr>0 and not(&syserr = 4)))) %then %do;
  %let hdds_err_flg=1;
  %final;
  /*Terminate SAS*/
  data _null_
    abort abend;
  run;
%end;

%mend chkErr;

%macro chkErr(checkpoint=);
APPENDIX C – FINAL MACRO

/*****************************************************************************************/
*                                                                                     *
* The %Final(), falls under the Footer section of the Error Handling Framework.        *
*                                                                                     *
* Macro Parameters: None                                                             *
*                                                                                     *
* Macros Called : 1) %Geterrors                                                      *
* 2) %Email                                                                          *
*                                                                                     *
/*****************************************************************************************/

%macro final;

/*Close the log and output external files*/
proc printto;
run;

/*If the Log content is being written to external file then call %getErrors to      *
parse the log file for Errors*/
%if ("&hdds_log_file" ne "") %then %do;
   %getErrors;
%end;

/*If Log content is not being written to external file*/
%if ("&hdds_log_file" = "") %then %do;
   /*If Error and Email Flag is on then send an email to owner of the SAS Process      *
   informing about the error condition*/
   %if (%eval(&hdds_email_flg=1 and &hdds_err_flg = 1)) %then %do;
      %let toid = &sysuserid.@healthdialog.com;
      %let msg = SAS Program Stopped After Checkpoint: &checkpoint;
      %email(to_addr1=&toid,subject=PROCESS &hdds_process_name caused a SAS Error,    *
         message="&msg");
   %end;

   /*If no Error and Email Flag is on then send an email to owner of the SAS       *
   Process informing about successful completion*/
   %else %if (%eval(&hdds_email_flg=1 and &hdds_err_flg = 0)) %then %do;
      %let toid = &sysuserid.@healthdialog.com;
      %let msg = No Critical Errors;
   %end;

%end;

/*If Error and Email Flag is on then send an email to owner of the SAS Error       *
informing about the error condition*/
%if (%eval(&hdds_email_flg=1 and &hdds_err_flg = 1)) %then %do;
   %let toid = &sysuserid.@healthdialog.com;
   %let msg = SAS Program Stopped After Checkpoint: &checkpoint;
   %email(to_addr1=&toid,subject=PROCESS &hdds_process_name caused a SAS Error,    *
      message="&msg");
%end;

/*If Error and Email Flag is on then send an email to owner of the SAS Error       *
informing about the error condition*/
%if (%eval(&hdds_email_flg=1 and &hdds_err_flg = 1)) %then %do;
   %let toid = &sysuserid.@healthdialog.com;
   %let msg = SAS Program Stopped After Checkpoint: &checkpoint;
   %email(to_addr1=&toid,subject=PROCESS &hdds_process_name caused a SAS Error,    *
      message="&msg");
%end;

/*If Error and Email Flag is on then send an email to owner of the SAS Error       *
informing about the error condition*/
%if (%eval(&hdds_email_flg=1 and &hdds_err_flg = 1)) %then %do;
   %let toid = &sysuserid.@healthdialog.com;
   %let msg = SAS Program Stopped After Checkpoint: &checkpoint;
   %email(to_addr1=&toid,subject=PROCESS &hdds_process_name caused a SAS Error,    *
      message="&msg");
%end;

/*If Error and Email Flag is on then send an email to owner of the SAS Error       *
informing about the error condition*/
%if (%eval(&hdds_email_flg=1 and &hdds_err_flg = 1)) %then %do;
   %let toid = &sysuserid.@healthdialog.com;
   %let msg = SAS Program Stopped After Checkpoint: &checkpoint;
   %email(to_addr1=&toid,subject=PROCESS &hdds_process_name caused a SAS Error,    *
      message="&msg");
%end;
%email(to_addr1=&toid,subject=PROCESS &hdds_process_name ran successfully,
    message="&msg");
%end;
%end;

%else %do;

/*If Error detected by %chkErr and Email Flag is on and there are errors parsed
   from the log file %geterrors then send an email to owner of the SAS Process
   informing about the error condition along with the SAS Error Messages in the
   Body of Email*/
%if (%eval(&hdds_email_flg=1 and &error_log ge 1 and &hdds_err_flg = 1)) %then %do;
  %let toid = &sysuserid.@healthdialog.com;
  %let msg = SAS Program Stopped After Checkpoint &checkpoint;
  %email(to_addr1=&toid,subject=PROCESS &hdds_process_name caused a SAS Error,
       message="&msg");
%end;

/*If Error not detected by %chkErr and Email Flag is on and there are errors
   parsed from the log file %geterrors then send an email to owner of the SAS
   Process with the SAS Error Messages in the Body of Email*/
%else %if (%eval(&hdds_email_flg=1 and &error_log ge 1 and &hdds_err_flg = 0)) %then %do;
  %let toid = &sysuserid.@healthdialog.com;
  %let msg = Non Critical Error occured;
  %email(to_addr1=&toid,subject=PROCESS &hdds_process_name caused a SAS Error,
       message="&msg");
%end;

/*If no Errors are detected and Email Flag is on then send an email to owner of
   the SAS Process informing about successful completion*/
%else %if (%eval(&hdds_email_flg=1 and &error_log = 0)) %then %do;
  %let toid = &sysuserid.@healthdialog.com;
  %let msg = No Errors;
  %email(to_addr1=&toid,subject=PROCESS &hdds_process_name ran successfully,
       message="&msg");
%end;
%end;
%mend final;
APPENDIX D – GETERRORS AND EMAIL MACRO

/***********************************************************************************/
*                                                                                 *
* The %getErrors and %Email are generic Macros being used in other HDAS Processes *
* as well.                                                                       *
*                                                                                 *
* The Geterrors Macro will read the the log file if saved and parse the log file  *
* for errors. These errors are then sent in the body of the Email., falls under   *
* the Footer section of the Error Handling Framework.                           *
*                                                                                 *
* The Email Macro being used in the Error Handling Framework gets the values for *
* its parameters from the %Final Macro. User is not required to provide the values* *
* for these parameters.                                                         *
*                                                                                 *
* Macro Parameters: Not Required                                                *
*                                                                                 *
* Macros Called : None                                                           *
*                                                                                 *
/***********************************************************************************/

%macro getErrors;

/*Extract Errors and Warnings from the Log File and stores them in their    *
corresponding data sets*/

data log(keep=logline) errors(keep=logline) warnings(keep=logline);
  infile "&hdds_log_file." missover;
  length logline $256 code $20;
  retain code ;
  input;
  if index(_infile_,"0D'x) then logline=scan(_infile_,1,"0D'x);
  else logline=_infile_
    logline = translate(logline," ','%');
  if index(logline,':')  then code=scan(logline,1,':');
  else if substr(logline,1,5) ne ' '  then code=scan(logline,1,' ');
  output log;
  if index(code,'ERROR') =1 and logline ne ' ' then output errors;
  if index(code,'WARNING') =1 and logline ne ' ' then output warnings;
run;

/*Error_Log Macro Variable captures number of error lines in the log file*/
proc sql noprint;
  select count(*)
    into :error_log
  from errors;
quit;

%mend getErrors;
%macro email(to_addr1=,
        to_addr2=,
        to_addr3=,
        to_addr4=,
        c_addr1=,
        c_addr2=,
        c_addr3=,
        c_addr4=,
        subject=,
        message=,
        attach_file1=,
        attach_file2=,
        attach_file3=);

filename outbox email;

data _null_;
file outbox
%do i = 1 %to 4;
  %if ("&&to_addr&i.." ne "") %then %do;
    to="&&to_addr&i..")
  %end;
%end;
%do i = 1 %to 4;
  %if ("&&c_addr&i.." ne "") %then %do;
    cc="&&c_addr&i..")
  %end;
%end;
subject="SASJOBS Automated Mail: &subject"

%let att_flg=0;
%do i = 1 %to 3;
  %if ("&&attach_file&i.." ne "") %then %do;
    %if (%eval(&att_flg=0)) %then %do;
      attach="&&attach_file&i..")
    %let att_flg=1;
    %else %do;
      "&&attach_file&i..")
    %end;
%end;
%end;
%if (%eval(&att_flg=1)) %then %do;
)
%end;
%end;
;
put &message;
run;

%mend email;