

SM06: Adapting the SAS Enterprise Guide Log Summary to Check Your LOG for You

Lawrence Heaton-Wright, IQVIA



10th–13th November
RAI Amsterdam
The Netherlands

EU 2019

The Clinical Data
Science Conference



- + Why Check Log Files?
- + Notes of Interest
- + Methods of Checking Log Files
- + EG/SAS Studio Log Summary
- + Customising the EG Log Summary
- + A Quick Recap

Why Check Log Files?

- To ensure that the program executes without errors or warnings
- Use the information provided within the LOG to check:
 - syntax and logical issues
 - timing information
 - SAS set up
- It's part of the overall quality control process
 - The LOG file must be “clean”
 - Free of errors (and unexplained warnings)
 - Ensuring any notes prohibited by company guidelines are not present
 - › These are what we have termed “Notes of Interest”

Notes of Interest

- Surely all SAS Log Notes are of “Interest”?
 - They are
 - But some are more interesting than others
- Assigning a libref successfully
 - This is expected behaviour from the study set up file
 - If not assigned, then an error will result
- Division by zero
 - This is not defined as an error or warning by SAS
 - But this is a Note of Interest
 - SAS is doing something by default because we haven’t defensively programmed around this issue
- Any log notes that fall into this category would be termed a Note of Interest

Methods of Checking Log Files

- Manually
 - Browsing the SAS interface Log window
 - Using an external text editor
- Automated
 - Return Codes
 - Using another tool/application
 - Using SAS macro(s)
 - EG/SAS Studio Log Summary

Methods of Checking Log Files - Manually

- This involves manually opening the LOG file
 - Switching to the SAS log window
 - Using an external text editor such as NotePad
- Search for errors, warnings and notes using the search functionality of the chosen viewer
 - Have to remember the search terms
 - It's repetitive and prone to inconsistency
 - Not really scalable for checking a lot of LOG files
- Best used for checking logic of the program
 - Checking the number of observations for input and output datasets
 - Checking the timing statistics of certain steps
 - Checking manually produced log notes

Methods of Checking Log Files - Automated

- These are performed as a post-processing activity following batch submission of SAS program(s)
- Involves checking/parsing LOG files using a set of specified strings
- SAS does not have error codes
 - Which would make life a lot easier
- SAS Return Codes
 - These are produced by SAS following a batch submission execution
 - Can be used to determine whether the program has produced warnings or errors
 - This allows a quick and easy check to identify programs that have not executed correctly
- Using another tool/application
 - For example UltraEdit programmable scripts
 - Fast but requires careful script writing to ensure correct notes picked out

Methods of Checking Log Files - Automated

- Using a SAS macro
 - Similar to using another tool/application
 - Except it's a SAS macro
 - Similar type of script required to identify notes of interest
 - Best used for scanning multiple log files
 - › Across an entire set of programs for instance
 - Results can be routed to any output destinations
 - › Excel/RTF/HTML – the choice is up to you
 - It can be used to read in the timing information for each step which can be analysed to identify long running steps

EG/SAS Studio Log Summary

The screenshot displays the SAS Enterprise Guide interface. The main window shows a SAS program with the following code:

```
79 59
80 60      DATA test;
81 61          x=5; y=7; z=9;
82 62          array vars[*] x y z;
83 63          do i=1 to dim(vars);
84 64              if vars[i] lt 6 then put _all_;
85
86          _22
87          200
88 ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, *, **, +, -, /,
89          IN, LE, LT, MAX, MIN, NE, NG, NL, NOT, NOTIN, OR, THEN, ^, ^=, |, ||, ~
90
91 ERROR 200-322: The symbol is not recognized and will be ignored.
92
93 65      end;
94 66      RUN;
95
96 NOTE: The SAS System stopped processing this step because of errors.
97 WARNING: The data set WORK.TEST may be incomplete. When this step was stopped there
98 WARNING: Data set WORK.TEST was not replaced because this step was stopped.
99
100
```

The Log Summary window is open at the bottom, showing a table of messages:

Description	Line	Affected Code
NOTE: The data set WORK.TEST has 1 observations and 3 variables.	71	run;
NOTE: Compressing data set WORK.TEST increased size by 100.00 percent.	72	run;
NOTE: DATA statement used (Total process time):	74	run;
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, *, **, +, -, /, <=, <...	88	if vars[i] lt 6 then put _all_;
ERROR 200-322: The symbol is not recognized and will be ignored.	91	if vars[i] lt 6 then put _all_;
NOTE: The SAS System stopped processing this step because of errors.	96	RUN;
WARNING: The data set WORK.TEST may be incomplete. When this step was stopped there	97	RUN;
WARNING: Data set WORK.TEST was not replaced because this step was stopped.	98	RUN;
NOTE: DATA statement used (Total process time):	99	RUN;

- This is supplied within the Enterprise Guide and SAS Studio environments
- Splits ERRORS, WARNINGS and NOTES into 3 tabbed views
- Clicking on each tab toggles between showing the Errors/Warnings/Notes in the Log Summary window
 - When the tab has the blue outline then these will be shown in the Log Summary window

EG/SAS Studio Log Summary

The screenshot displays the SAS Enterprise Guide interface. The main window shows a SAS program with the following code:

```
NOTE: DATA statement used (Total process time):  
real time      0.00 seconds  
cpu time       0.00 seconds  
  
59  
60      DATA test;  
61          x=5; y=7; z=9;  
62          array vars[*] x y z;  
63          do i=1 to dim(vars);  
64              if vars[i] lt 6 then put _all_;  
65  
66          end;  
67      RUN;
```

The Log Summary window is open, showing a list of messages. The 'Notes' tab is selected, displaying the following messages:

Description	Line	Affected Code	Log
ERROR: TESTING IMPOSSIBLE DATA POINT LOG MESSAGE	53	run;	58
ERROR: TESTING IMPOSSIBLE DATA POINT LOG MESSAGE	54	run;	58
ERROR 22-322: Syntax error, expecting one of the following: !, , &, *, **., +, -, /, ;, <, <=, <>, =, >, >=, AND, EQ, GE, GT, IN, LE, LT, MAX, MIN, NE, NG, NL, NOT, NOTIN, OR, THEN, ^, ^=, , , ~, ~=.	88	if vars[i] lt 6 then put _all_;	64
ERROR 200-322: The symbol is not recognized and will be ignored.	91	if vars[i] lt 6 then put _all_;	64
WARNING: The data set WORK.TEST may be incomplete. When this step was stopped there were 0 observations and 4 variables.	97	RUN;	66
WARNING: Data set WORK.TEST was not replaced because this step was stopped.	98	RUN;	66

- This is supplied within the Enterprise Guide and SAS Studio environments
- Splits ERRORS, WARNINGS and NOTES into 3 tabbed views
- Clicking on each tab toggles between showing the Errors/Warnings/Notes in the Log Summary window
 - When the tab has the blue outline then these will be shown in the Log Summary window
 - By clicking on the Notes tab this has been toggled from “showing” to “not showing”

EG/SAS Studio Log Summary

The screenshot displays the SAS Enterprise Guide interface. The main window shows the SAS code being executed, with line numbers 79 to 97. The code includes a DATA statement, an array statement, and a loop. The Log Summary window is open, showing the Log window with the following content:

```
79 59
80 60      DATA test;
81 61          x=5; y=7; z=9;
82 62          array vars[*] x y z;
83 63          do i=1 to dim(vars);
84 64              if vars[i] lt 6 then put _all_;
85
86              _
87              22
88              200
89 ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, *, **, +, -, /,
90          IN, LE, LT, MAX, MIN, NE, NG, NL, NOT, NOTIN, OR, THEN, ^, ^=, |, ||, ~
91 ERROR 200-322: The symbol is not recognized and will be ignored.
92
93 65      end;
94 66      RUN;
95
96 NOTE: The SAS System stopped processing this step because of errors.
97 WARNING: The data set WORK.TEST may be incomplete. When this step was stopped there
98
99 WARNING: Data set WORK.TEST was not replaced because this step was stopped.
```

The Log Summary window is at the bottom, showing a table of log messages:

Description	Line	Affected Code
NOTE: The data set WORK.TEST has 1 observations and 3 variables.	71	run;
NOTE: Compressing data set WORK.TEST increased size by 100.00 percent.	72	run;
NOTE: DATA statement used (Total process time):	74	run;
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, *, **, +, -, /, <, <=, <...	88	if vars[i] lt 6 then put _all_;
ERROR 200-322: The symbol is not recognized and will be ignored.	91	if vars[i] lt 6 then put _all_;
NOTE: The SAS System stopped processing this step because of errors.	96	RUN;
WARNING: The data set WORK.TEST may be incomplete. When this step was stopped there	97	RUN;
WARNING: Data set WORK.TEST was not replaced because this step was stopped.	98	RUN;
NOTE: DATA statement used (Total process time):	99	RUN;

- This is supplied within the Enterprise Guide and SAS Studio environments
- Splits ERRORS, WARNINGS and NOTES into 3 tabbed views
- Clicking on each tab toggles between showing the Errors/Warnings/Notes in the Log Summary window
 - When the tab has the blue outline then these will be shown in the Log Summary window
- Double clicking on the WARNING in the Log Summary moves the pointer in the Log window to the line where the log note appears

Customising the EG Log Summary

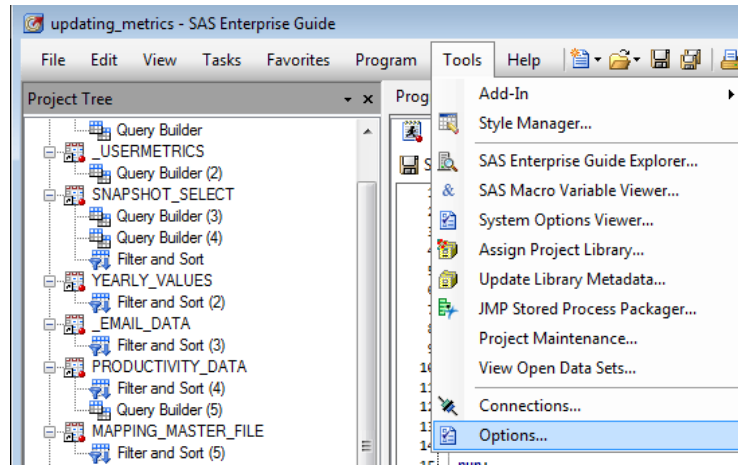
- At IQVIA we migrated to a SAS Grid environment
 - Previously used SAS servers utilising them as a giant PC-SAS environment
 - So we were used to using SAS Display Manager (DMS)
- Moving to a Grid environment meant using EG or SAS Studio
- The EG (and Studio) Log Summary is a leap forward from PC-SAS Log windows
- We already had other LOG checking tools
 - UltraEdit customised scripts
 - A SAS macro suite

- Was there anything we could do in EG?

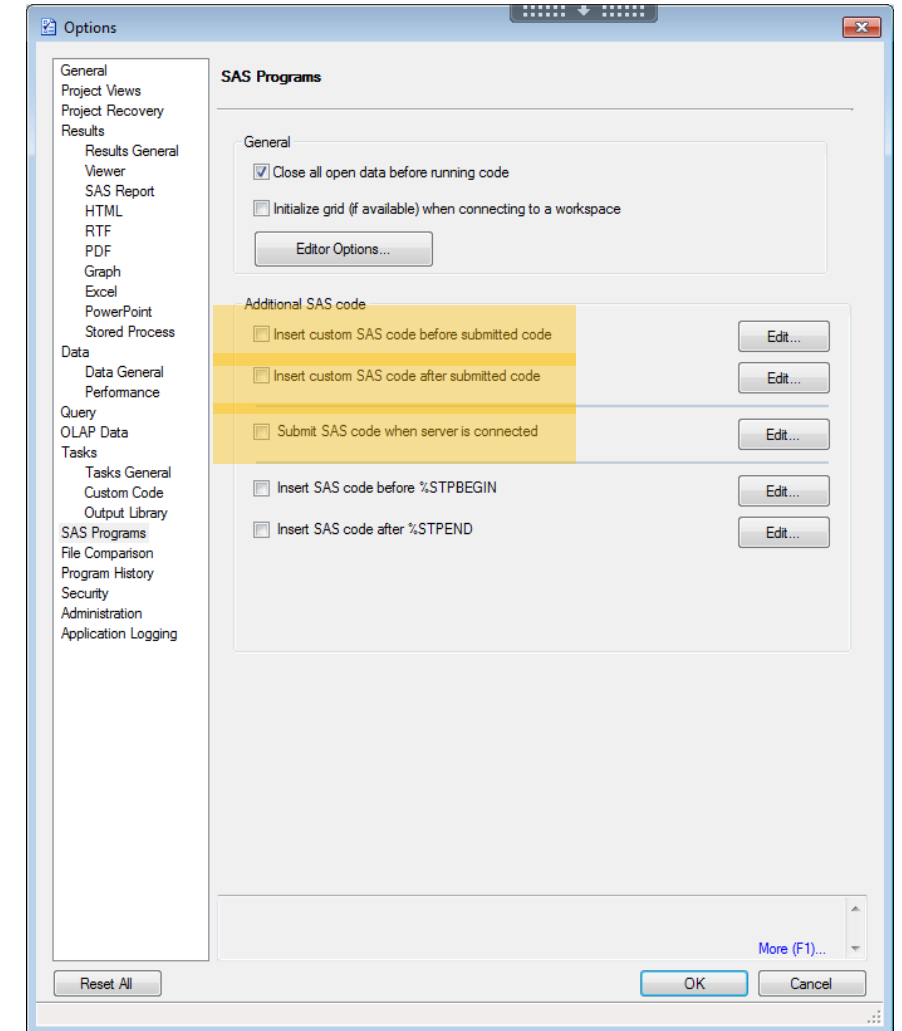
Customising the EG Log Summary

- Investigation of the EG menus - Tools>Options>SAS Programs – showed 3 potentially useful options that allow code to be submitted at specified points in the submission process

- On server connection
- Prior to submitted code
- After submitted code

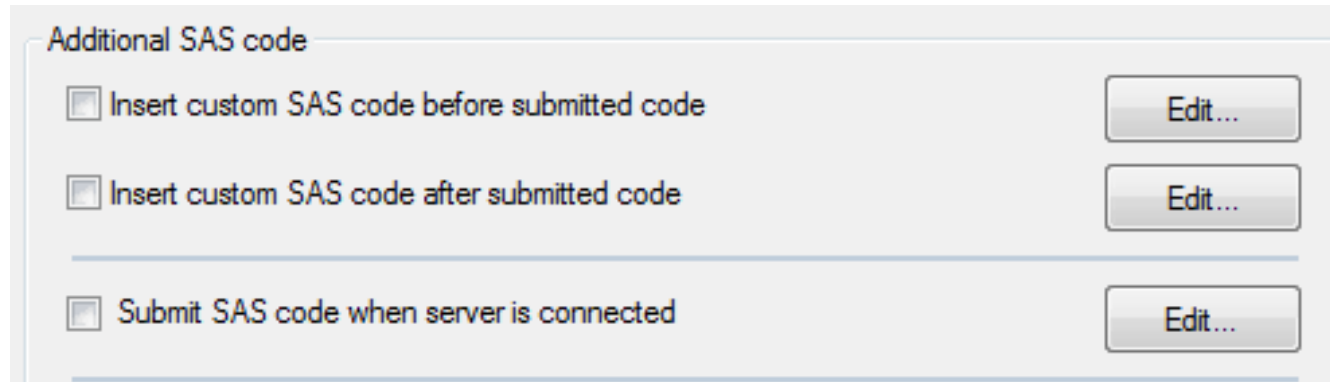


- These options effectively allow us to customise the EG “wrapper” code which EG uses to submit code to the SAS Grid node
- We developed a method of using the logic of our existing log checking SAS macro code, sending the results to the EG Log Summary Window to be highlighted as Warnings or Notes as appropriate



Customising the EG Log Summary

- Each of these checkboxes has an “Edit...” button
- These edit buttons open a SAS programming window allowing you to insert SAS code
- This code is submitted at the point of submission as described

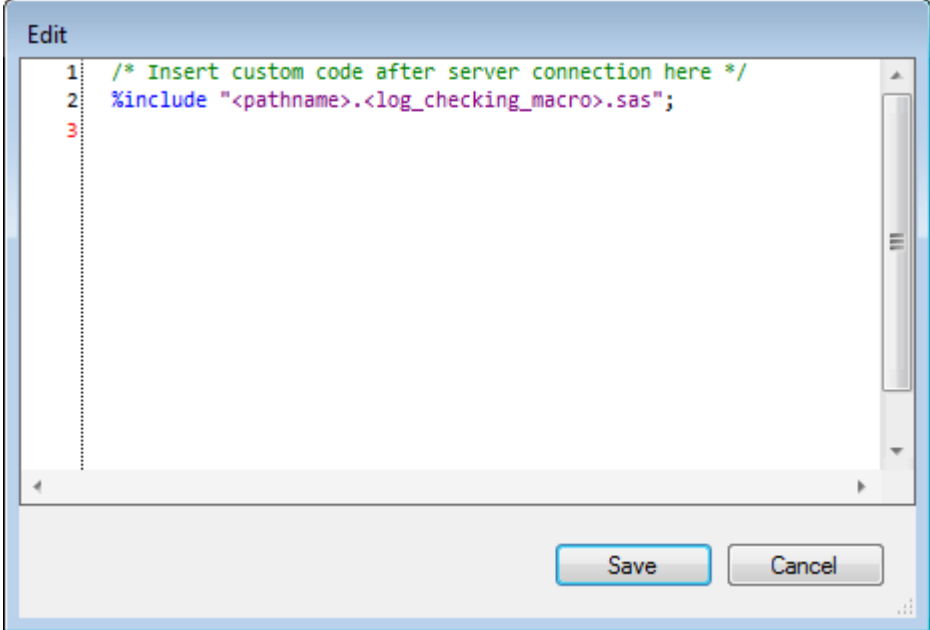


The screenshot shows a dialog box titled "Additional SAS code". It contains three rows, each with a checkbox and an "Edit..." button. The first row has the checkbox "Insert custom SAS code before submitted code". The second row has the checkbox "Insert custom SAS code after submitted code". The third row has the checkbox "Submit SAS code when server is connected".

Option	Edit...
☐ Insert custom SAS code before submitted code	Edit...
☐ Insert custom SAS code after submitted code	Edit...
☐ Submit SAS code when server is connected	Edit...

Submit SAS code when server is connected

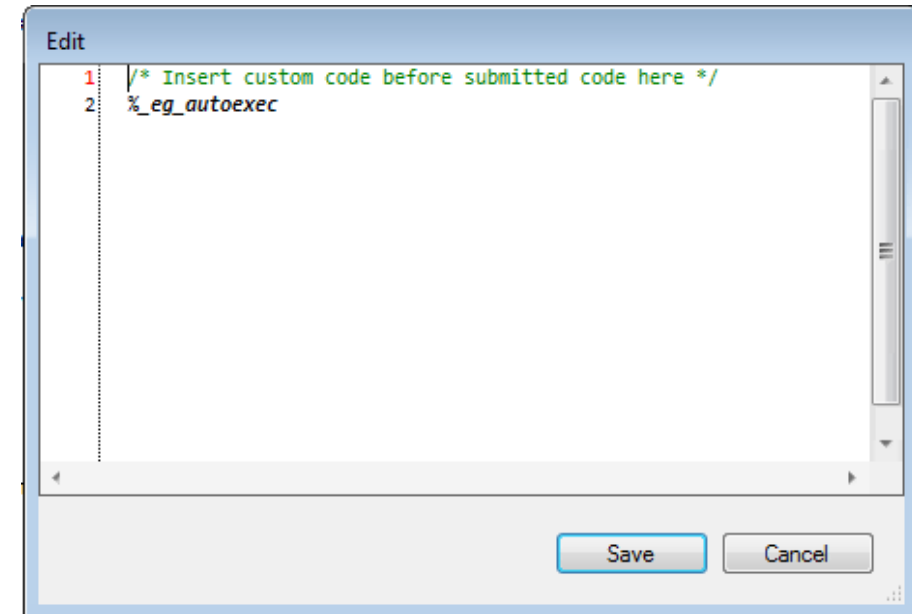
- The code entered here %INCLUDEs the SAS program that contains the LOG re-routing and parsing macros which perform the enhanced log scanning
- This section can be used for other macros or code to submit on server connection
 - At IQVIA we have a standard study set-up code that we execute when a program is submitted via batch mode
 - We have EG-specific code included here to replicate the batch submission process (macro variables, librefs, filerefs, etc.)

A screenshot of a software window titled "Edit". The window contains a text editor with three lines of SAS code. Line 1 is a comment: "/* Insert custom code after server connection here */". Line 2 is a macro call: "%include "<pathname>.<log_checking_macro>.sas";". Line 3 is empty. The code is color-coded: green for comments, blue for macro names, and black for text. At the bottom right of the window are two buttons: "Save" and "Cancel".

```
1  /* Insert custom code after server connection here */
2  %include "<pathname>.<log_checking_macro>.sas";
3
```

Insert custom SAS code before submitted code

- Invoke one of the macros included in the previous step
 - In our case called `_EG_AUTOEXEC`
 - This includes code to include our study set-up programs (excluded from this example)
 - Re-routing of the LOG window to a folder within the WORK library using PROC PRINTTO

A screenshot of an SAS 'Edit' window. The window has a title bar that says 'Edit'. Inside, there is a text editor with two lines of code. Line 1 is a comment: `/* Insert custom code before submitted code here */`. Line 2 is a macro call: `%_eg_autoexec`. The code is color-coded: the comment is green and the macro call is black. At the bottom right of the window, there are two buttons: 'Save' and 'Cancel'.

```
1  /* Insert custom code before submitted code here */
2  %_eg_autoexec
```

Insert custom SAS code before submitted code - detail

```
]%macro _eg_autoexec;
  %*****
  %* Turning off macro options
  %*****;
  %let __origopts = %sysfunc(getoption(mprint)) %sysfunc(getoption(symbolgen)) %sysfunc(getoption(mlogic));
  options nomprint nosymbolgen nomlogic;

  %*****
  %* Change log destination prior to submitting code
  %*****;
  %* Getting the WORK directory;
  %global __workpath;
  %let __workpath = %sysfunc(pathname(work));

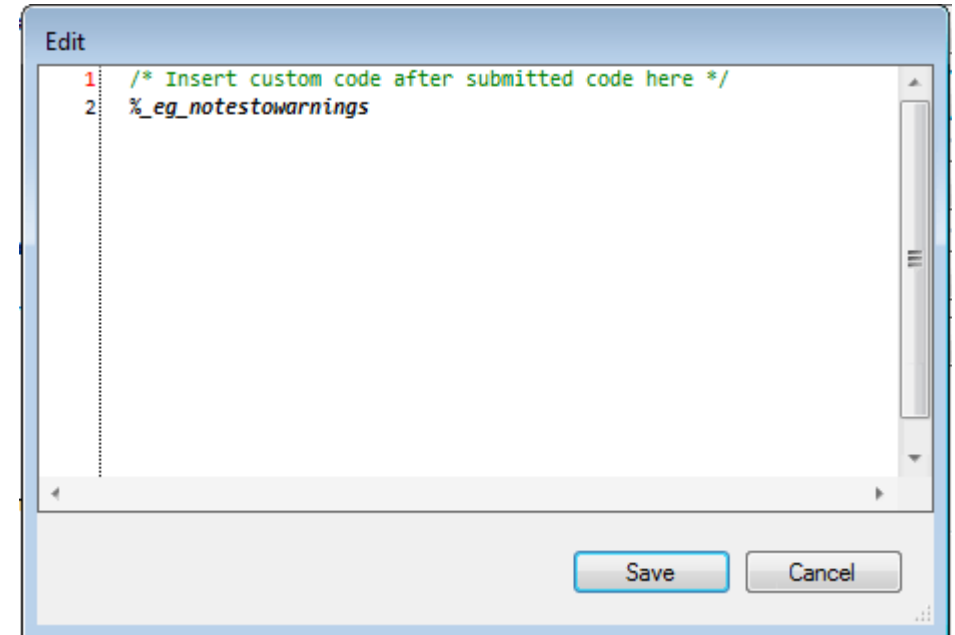
  proc printto log = "&__workpath.\_EG_TempLog_&sysuserid..log" new;
  run;

  %*****
  %* Restore original options
  %*****;
  options &__origopts.;
%mend _eg_autoexec;
```

- The WORK library is a physical file location
- Use the PATHNAME function to ascertain the actual folder path
- Use a PROC PRINTTO to re-route the LOG file to this physical WORK area

Insert custom SAS code after submitted code

- Invoke one of the macros from the server connection %INCLUDE section
- In our case called _EG_NOTESTOWARNINGS
 - This resets the LOG re-routing to the LOG window
 - It reads in the temporary LOG file
 - Converts Notes of Interest to WARNING
 - Converts other Notes to INFO
 - › These are notes that we expect to appear (such as librefs correctly assigned)
 - › They are demoted to information so they are available but do not clog up the log summary window
 - By using a PUT statement the modified LOG file notes are scanned by the EG Log Summary and “new” WARNINGS appear in the Warnings section of the log summary



Some more detail on the Notes to Warnings macro

- The first step is to turn off the LOG re-routing
- Read in the temporary LOG file using a DATA _NULL_
 - We don't want to create extra datasets
- The next step is to convert NOTES (and ERRORS) from the format "NOTE 484-185:" to "NOTE:"
 - This code could be improved a bit by using PRX functions

```
*****  
%* Convert notes of interest to warnings after submitting code  
*****;  
proc printto log = log;  
run;  
  
data _null_;  
  length line $32767;  
  infile "&__workpath.\_EG_TempLog_&sysuserid..log" length = __length;  
  input @1 line $varying32767. __length;  
  
  %* Converting NOTE xxx-xxx to NOTE;;  
  if kscan(line, 1, " ") in ("ERROR", "WARNING", "NOTE", "INFO") then do;  
    length code $10;  
    code = kscan(line, 2, " ");  
    if kindex(code, ":") = klength(code) and not kverify(kstrip(code), "1234567890-:")  
      then line = kscan(line, 1, " ") || ":" || ksubstr(line, kindex(line, ":") + 1);  
  end;
```

Some more detail on the Notes to Warnings macro

- Then we convert the Notes of Interest we want to WARNINGS
 - This is where error/note codes would be very handy!
- Then the other notes are converted to INFO
 - This means that Notes that we haven't dealt with (promoted or demoted) appear as Notes
- The PUT statement then returns this to the LOG window
- The EG Log Summary does the rest

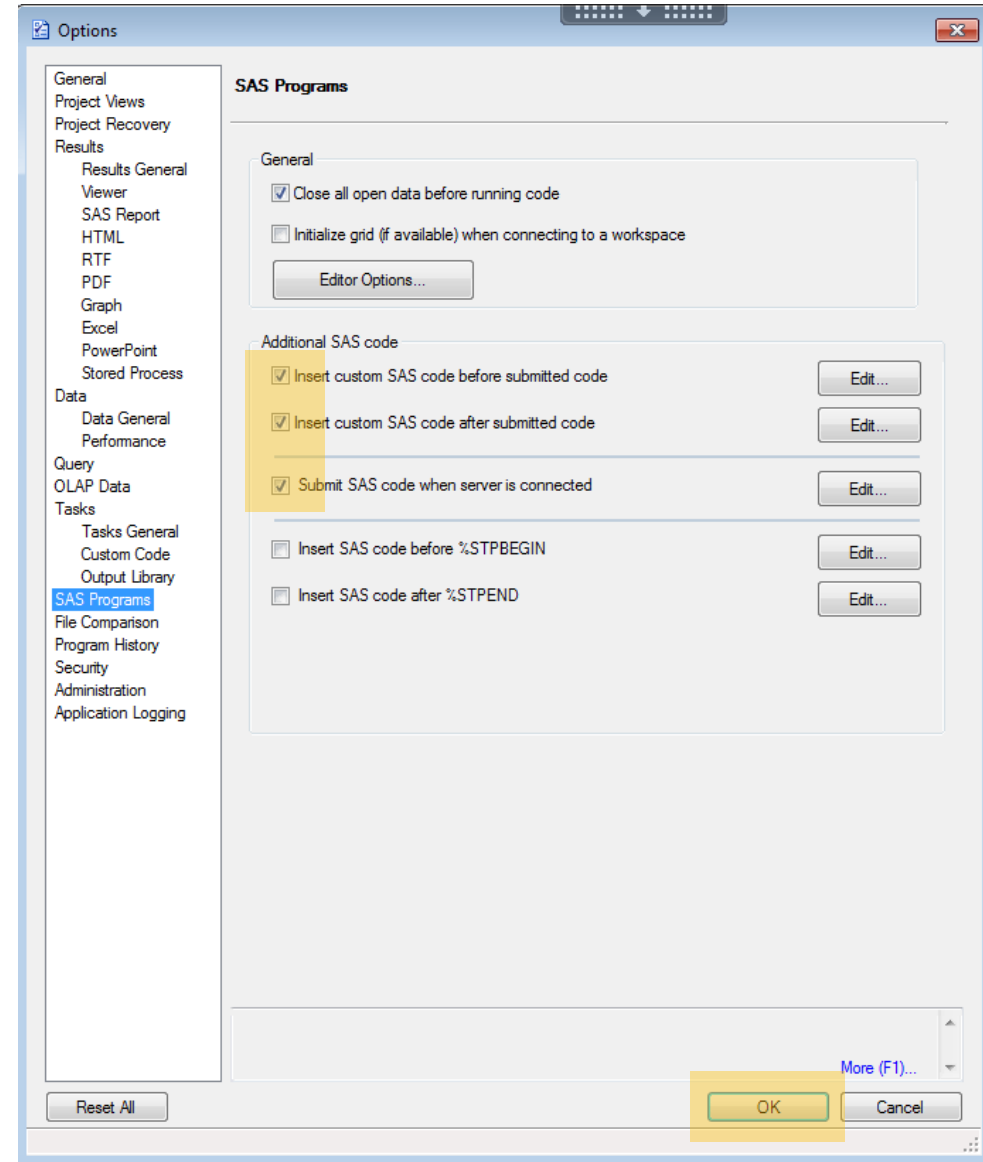
```
/* Converting NOTE and INFO of interest into warnings;
if ksubstr(line, 1, 6) in ("NOTE: ", "INFO: ") then do;
  if
    kindex(kupcase(line), "UNINITIALIZED") or
    kindex(kupcase(line), "INVALID") or
    kindex(kupcase(line), "DIVISION BY ZERO") or
    ...
    kindex(kupcase(line), "SOME GRAPH LEGENDS HAVE BEEN DROPPED DUE TO SIZE CONSTRAINTS")
  then line = "W" || "ARNING: " || ksubstr(line, 7);
end;

else if kindex(kupcase(line), "THE DATA SET") or
  kindex(kupcase(line), "NOTE: LIBRARY") or
  kindex(kupcase(line), "NOTE: LIBREF") or
  ...
  kindex(kupcase(line), "NOTE: FILEREF") or
  kindex(kupcase(line), "NOTE: TABLE") or
  kindex(kupcase(line), "NOTE: SQL")
  then line = catx(' ', 'INFO:', ksubstr(line, 7));

/* Print modified log;
put line;
run;
```

Customising the Log Summary

- Now we've added the code to the edit buttons
- Tick the check boxes
- Click on OK
- Exit EG
 - The changes for won't take effect until a new session is opened
- Open a new EG session
- Run a program and check the updated Log Summary



EG Log Summary Customisation – in action

- The Notes to Warnings macro will convert Notes of Interest to Warnings
 - The NOTE 484-185: will be promoted to a Warning
 - This makes it much easier to spot in the Log Summary as warnings are (hopefully) much rarer than Notes
- Some notes are demoted to Info so they won't appear in the Log Summary at all but will still appear in the Log

Log Summary		
 Errors (0)	 Warnings (0)	 Notes (4)
Description		Line
 NOTE 484-185: Format _NOFMT was not found or could not be loaded.		12
 NOTE: The data set WORK.TEST has 1 observations and 3 variables.		16
 NOTE: Compressing data set WORK.TEST increased size by 100.00 percent.		17
 NOTE: DATA statement used (Total process time):		19

Log Summary		
 Errors (0)	 Warnings (1)	 Notes (10)
Description		Line
 WARNING: Format _NOFMT was not found or could not be loaded.		50

EG Log Summary Customisation – in action

- Some notes are demoted to Info so they won't appear in the Log Summary at all but will still appear in the Log
- The notes that appear here are from the wrapper code or are notes that haven't been promoted or demoted

```
-- MPRINT(GJOBS): title my Grid jobs today (completed or ongoing within the last hour) ,
395 MPRINT(GJOBS): proc print data=zsasgsub.jobs94 width=minimum heading=horizontal noobs label;
396 MPRINT(GJOBS): var job_id user status month day time jobname ex_server;
397 MPRINT(GJOBS): run;
398
399 INFO: There were 1 observations read from the data set ZSASGSUB.JOBS94.
400 INFO: PROCEDURE PRINT used (Total process time):
401 real time          0.00 seconds
402 cpu time           0.00 seconds
403
404
405 MPRINT(GJOBS): title;
406 MPRINT(GJOBS): options nospool;
407 37
408 38
409 39 /* Insert custom code after submitted code here */
410 40 %_eg_notestowarnings
411 41
412 42 %_eg_hidenotesandsources;
413 MPRINT(_EG_NOTESTOWARNINGS): options nomprint nosymbolgen
414 NOTE: 367 records were read from the infile "f:\sas_work\TD28636_IEEDC-SASGCPN02\Pr2\_EG_TempLog_q633753.log".
415 The minimum record length was 0.
```

Log Summary

Errors (0) Warnings (0) Notes (5)

Description	Line	Affected Code
NOTE: PROCEDURE PRINTTO used (Total process time):	20	%gridInfo; *** Shows BHOSTS/LSLOAD information ***;
NOTE: The infile "f:\sas_work\TD28636_IEEDC-SASGCPN02\Pr2_EG_TempLog_q...	26	%gridInfo; *** Shows BHOSTS/LSLOAD information ***;
NOTE: _EG_IncBiosetup did not run because [&root] already exists.	39	%gridInfo; *** Shows BHOSTS/LSLOAD information ***;
NOTE: 367 records were read from the infile "f:\sas_work\TD28636_IEEDC-SASGCPN0...	414	%gridInfo; *** Shows BHOSTS/LSLOAD information ***;
NOTE: DATA statement used (Total process time):	417	%gridInfo; *** Shows BHOSTS/LSLOAD information ***;

A Quick Recap

- There are several methods that can be used to check SAS LOG files
- Manual
 - Prone to missing issues
 - Best suited to finding logical errors
- Automated using SAS return codes
 - Quick but only shows whether a program execution generates Errors, Warnings or Notes only
- Automated using another application
 - Quick but requires a detailed script to be written
- Automated using a SAS macro
 - Allows precision in finding log issues and can send output to many destinations
 - Slower as the log file(s) have to be read into SAS datasets
- Using the EG Log Summary window
 - Summarised view of Errors, Warnings and Notes
 - Still requires manual searching of notes

A Quick Recap

- Customising the EG Log Summary window allows the user to concentrate on the SAS Log Notes of Interest to them
 - Use the algorithm from an existing SAS log checking macro to promote Notes of Interest to warnings – this forms the basis of the macros used
 - Use the 3 EG options to execute code at session start-up and before/after code submission
 - › The SAS code inserted here allows the execution of log checking macros
 - › The modified log notes are sent back out [PUT statement] and the EG log summary treats these as a “normal” log
 - Doing this highlights these issues to the programmer instantly
 - Other log notes which are demoted to information (while still being useful) do not clog up the log summary window

A Quick Recap

- Identifying SAS log issues during program development saves a lot of time
 - There should be less log issues picked up during a post-processing log file check
 - › Post-processing checks are still required
 - This should mean less re-work, which is always a good thing
 - The log window can still be checked manually so that logical issues can be identified
 - This method combines the best aspects of automation and manual review
- All the LOG checking methods discussed have their place – pick the best method for what you're doing at the time



Any questions?

