SAS® 9.1.3 Qualification Tools User's Guide

Copyright Notice

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® 9.1.3 Qualification Tools User's Guide*, Cary, NC: SAS Institute Inc., 2007.

Table of Contents

Introduction

There are two qualification tools available with SAS 9.1.3—the SAS Installation Qualification Tool (SAS IQ) and the SAS Operational Qualification Tool (SAS OQ). These tools will assist you in qualifying the use of the SAS software in regulated industries. Together, these tools support the qualification aspect of the essential migration, integration, and verification processes customers need to move from previous versions of SAS to later releases.

The SAS IQ assists you in demonstrating the SAS System has been installed and maintained to the manufacturer's specifications. SAS IQ verifies the integrity of each file in the SAS System 9 and provides the customer a set of reports detailing the results. SAS IQ is supported on all Windows, UNIX, and OpenVMS platforms for SAS 9.1.3. See ‘SAS Installation Qualification Tool” on page 3 for a detailed description.

The SAS OQ assists you in demonstrating the SAS System is operational. SAS OQ uses SAS programs provided by the component development groups and will execute, process, and report the program results. SAS OQ is supported on all Windows and UNIX platforms for SAS 9.1.3. See “SAS Operational Qualification Tool” on page 7 for more information.

SAS Installation Qualification Tool

Overview

The SAS Installation Qualification Tool (SAS IQ) validates a SAS System installation by verifying that each installed file is correct. A report is generated detailing all file results. This determination is made using the md5 algorithm to create a value for each file.

Windows

Running the Installation Qualification Tool

There are two ways to run the SAS IQ on a successful SAS 9.1.3 installation:

- From the start menu as follows:
  `start → programs → SAS → sas 9.1 utilities → SAS Installation Qualification tool`

or

- Open a command line in DOS and type the following command:

  `c:\><SASROOT>\sastest\sasiq.exe -OutputPath <dir>`

  In this command, <*SASROOT*> is the location where the SAS System was installed, and <*dir*> is an empty directory.

Both of these methods require an output directory, which must be empty.

Output

The SAS IQ generates XML data files to hold the raw results of the validation process. The SAS System is used to translate the XML data into PDF and HTML reports. The PDF and HTML reports are identical. The PDF can be used to create hardcopies of the resulting validation. The HTML is easier to browse.

For the Windows platform, XSL translation files are also provided. These allow another means of generating HTML from the XML data file. However, the XSL translation is done dynamically.

In all cases, the output report is organized first at the SAS System component level, and then by each component's files.

OutputPath option

The following table describes the structure of the directory used in the `-OutputPath` option:

File Name	Description
`Data`	Directory containing the raw validation data for each component in XML format
`html`	Directory containing HTML files for each component
`autoexec.sas`	SAS file used in producing PDF and HTML output from XML data
`index.xml`	XML index file for all component summary
`index.xsl`	XML style sheet
`media.xsl`	XML style sheet
`sasiq.htm`	HTML file containing index to summary and component information
`sasiq.log`	SAS log file created from running the autoexec.sas program
`sasiq.pdf`	PDF file containing all summary and component information
`SASIQindex.map`	Intermediate file used to output XML to PDF and HTML
`SASIQprod.map`	Intermediate file used to output XML to PDF and HTML
`sasiqtoc.htm`	HTML file used to display component information

UNIX

Method 1: SAS Installation Representative Running the Installation Qualification Tool

The SAS Installation Representative can run the SAS IQ at any time by following the steps below:

1. Run **SAS Setup** from `!SASROOT/sassetup`. Make sure you have the appropriate privilege to update files in SASROOT.

2. Select **Run Setup Utilities** from the SAS Setup Primary Menu.

3. Select **Perform SAS System Configuration**.

4. Select **Run the SAS Installation Qualification Tool**.

5. Check the installation log file for a summary of the results or review the detailed report in the .xml file that SAS Setup lists on the screen.

This will create validation reports and logs in the `!SASROOT/install/admin/ validate_`*`date.version`* directory.

Method 2: User Running the SAS Installation Qualification Tool

A user of SAS can run the SAS IQ at any time without needing write access to the SASROOT directory by passing the –validate and –valoutdir command flags to SAS Setup. These command flags redirect all log files and output to the specified directory. The user must have write access to the specified directory. An example of the command is:

```
$ !SASROOT/sassetup –validate –valoutdir <directory>
```

where *<directory>* is a directory to which the user has write access.

Reviewing the Results of the SAS Installation Qualification Tool

After you run the SAS IQ, a `validate_date.version` directory is created either in the `!SASROOT/install/admin` directory or in the output directory you specified, based on how you invoked the SAS IQ (see the above two methods for invoking the SAS IQ). You can review the results of the SAS IQ by opening the `sasiq.htm` file in a Web browser or by opening the `sasiq.pdf` file in Adobe Reader.

The contents of the validate directory are described in the following table:

File Name	Description
`Data/`	Directory containing XML files for each component
`html/`	Directory containing HTML files for each component
`autoexec.sas`	SAS file used in producing PDF and HTML output from XML data
`index.xml`	XML index file containing installation validation summary for installed software
`index.xsl`	XML style sheet
`media.xsl`	XML style sheet
`sasiq.htm`	HTML file containing installation validation summary for installed software
`sasiq.pdf`	PDF file containing installation validation summary for installed software
`SASIQindex.map`	Intermediate file used to output XML to PDF and HTML
`SASIQprod.map`	Intermediate file used to output XML to PDF and HTML
`sasiq.log`	SAS log file created from running the autoexec.sas program
`sasiqtoc.htm`	File containing HTML output table of contents

The SAS IQ validates a SAS System installation by verifying that each installed file is correct. A report is generated detailing all file results. The status determination is made using the md5 algorithm to create a value for each file.

SAS Operational Qualification Tool

This chapter describes the tools used to run operational tests of an installed image of SAS 9.1.3 at your site. These tools and testware are shipped on SAS 9.1.3 media.

Supplied Tool Set

The following tools are included in SAS 9.1.3.

- SAS Operational Qualification Tool (SAS OQ)—the program that invokes the operational tests. It accepts various command-line arguments (documented below). It runs the tests, validates the output from the test, and produces the report.
- IVFND—the "Filter 'N' Diff" program. The test output and benchmarks need to be filtered before they are compared in order to remove spurious differences such as date/time values and pathnames. IVFND runs the appropriate filters and difference tool to validate an output file.
- PED—a Perl version of the UNIX `sed` command that provides the filtering capability to IVFND.
- IVDIFF (`sasdiff.exe`)—a difference tool that has some special options to control the difference.

User Environment

Generally, the user environment includes the customer tests, SAS supplied tests, test output area, and the SAS image to be tested. The customer tests must be organized in the same format as the SAS supplied tests. The SAS supplied tests are segregated by component in subdirectories. Each component subdirectory contains the SAS programs and the test tables. A directory for the test output will be needed. Attempting to write to a directory with existing test data will cause an error message.

Directory Structure:

UNIX

```
SASROOT - /usr/local/sas/
SAS Operational Qualification Tool - /usr/local/sas/sastest/sasoq.sh
SAS Operational Qualification Tool support files - /usr/local/sas/sastest
Component content - /usr/local/sas/sastest/<component>
```

Windows

```
SASROOT - C:\Program Files\SAS\SAS 9.1\
SAS Operational Qualification Tool and support files -
C:\Program Files\SAS\SAS 9.1\sastest\
Component content - C:\Program Files\SAS\SAS 9.1\sastest\<component>
```

Invocation

The general command line invocation for Windows is:

```
<SAS Operational Qualification Tool path>sasoq.exe –tables *:<component> ...
```

The general command line invocation for UNIX is:

```
<SAS Operational Qualification Tool path>sasoq –tables *:<component> ...
```

This command line would cause the SAS OQ to execute all the available tests for each component named on the command line, and put the test output and data in a new subdirectory `ftt_<YYYYMMDD>.000` of the current directory. The trailing `.000` would be incremented for tool invocations on the same day.

To specify the testware location for customer tests:

```
<SAS Operational Qualification Tool path>sasoq.exe –testware <path to testware>
-tables *:<subdirectory> ...
```

This command line would only run the tests in the –testware path.

To specify an output path for the generated data files:

```
<SAS Operational Qualification Tool path>sasoq.exe –tables *:base *:ets *:or
    *:qc *:stat
-outdir c:\public\mydata
```

This command line, for Windows, will run the tests in the `base`, `ets`, `or`, `qc`, and `stat` directories provided by SAS and put the data file in the path `c:\public\mydata`. If the subdirectory `mydata` does not exist, it will be created.

The tool can provide more detail in the screen output by using the verbose option:

```
/usr/local/sas/sasoq.sh –tables *:ets –outdir /users/guest/mydata –verbose
```

This UNIX command line will run the ETS tests, put the data files in a local `mydata` directory, and provide the sas command line used to run the test, any processing statements, and the results of the test, such as:

```
Running test ets:tstets:tstari01...
Running: sas  -sysin "/usr/local/sas/sastest/ets/tstari01.sas"  -autoexec
"/usr/local/sas/sastest/base/assert.sas"  -nodate -nostimer -ls 78 -ps 60 -
noovp -nosyntaxcheck
sasoq:  The command returned the proper value (0).
Processing Results:
```

The quiet option will suppress all non-error output.

```
<SAS Operational Qualification Tool path>/sasoq.sh –tables *:ets –outdir
/users/guest/mydata –quiet
```

This UNIX command line will produce no output from the test execution until all tests are run. The tool will continue to produce the message directing the user to the appropriate path for the data files.

Automation

The SAS OQ can be run in an automated manner to allow for scheduled operational tests on a production or test SAS 9.1.3 image. To establish the SAS OQ as a scheduled task on SAS 9.1.3-supported Windows operating systems, use the following steps:

1. Create a script similar to the example below.
2. To create a scheduled task:
   - Find the Scheduled Task interface. On newer Windows systems this is found under Control Panel
   - Use the wizard to create the scheduled task
   - In the Task panel, execute the script in the example using the wscript interface:
     `wscript.exe "c:\sasiq\runoq.vbs"`
   - In the Schedule panel select the appropriate frequency
   - Press OK to create the Scheduled Task
3. Review the tool output after each invocation.

To establish the SAS OQ as a crontab task on SAS 9.1.3 supported UNIX operating systems, use the following steps:

1. Create a script similar to the UNIX example below.
2. To execute the UNIX example enter the following into a crontab entry:

```
# run SAS 9.1 SAS Operational Qualification Tool process
# at 09:05am Weekdays
05 09 * * 1,2,3,4,5 "/users/guest/bin/runoq > /users/guest/mydata/runoq.log"
```

3. Review the tool output after each invocation

Windows Example:

```
'* runoq.vbs - run the SAS Operational Qualification Tool using all the
   component tests
'* First edition

'* run this script only on the local machine
strComputer = "."

oqpath = "\sasoq\mydata"

'* create the string to use for executing the tool
initoq = "c:\program files\sas\sas 9.1\sastest\sasoq.exe -tables *:base *:dmine
*:ets *:graph *:hpf *:iml *:insight *:irp *:lab *:or *:qc *:stat -outdir " &
oqpath

'* execute the tool
Set WshShell = WScript.CreateObject("WScript.Shell")
Set oExec = WshShell.Exec(initoq)
```

UNIX Example:

```
#!/bin/sh
#runoq – execute the SAS Operational Qualification Tool

/usr/local/sas/sastest/sasoq.sh –tables *:base *:dmine *:ets *:graph *:hpf
*:iml *:insight *:irp *:lab *:or *:qc *:stat –outdir /users/guest/mydata
```

Output Report

The SAS OQ stores the raw output data in XML files. SAS 9.1.3 is used to translate the raw XML data into PDF and HTML reports. The PDF and HTML reports are identical, but each has its own purpose—the PDF should be used to generate a hardcopy of the final results, while the HTML is a more convenient means of interactively examining the output.

The output report separates the tests by component and then by test table. The test summary lists all data pertinent to the test execution—command line, return codes, output files compared against benchmarks (if any), etc.

SAS Operational Qualification Tool Table Language

The primary input files for the SAS OQ are test tables (stored in files with a `.tab` extension). These files define which tests to run, how to run the tests, and how to determine if the tests worked correctly.

Table Language General Rules

The following are basic rules for test table files:

- The SAS OQ generally ignores white space (spaces or line breaks).
- Comments are delimited by /* and */. They can span multiple lines and appear anywhere in a line. These comments work the same as C comments.
- Statement arguments fall into one of the following groups:
  - A **list** is a group of items in any order separated by commas. An example of a list is the set of options passed to the program being tested. The order of such options is irrelevant to the operation of the SAS OQ. (The program being tested might require them to be in a particular order, but the SAS OQ does not).
  - A **structure** is a group of items in a specific order separated by colons. An example of a structure is a file specification. The elements must be in a particular order, so that the SAS OQ can tell which part is the file name, which part is the component name, etc.
- Statements are processed in the order in which they occur in the test block, so the statement to run the test must be before the statements to process the results.
- Tables are made up of comments, `&set` commands, `&unset` commands, and test blocks. Each test block starts with the block identifier, `&test`, and contains a set of commands—a single `&run` followed by `&process` and/or `&rc`. Within the commands you can optionally use the macros `&infile, &outfile` and `&resfile`.

Block Identifier

The only valid block identifier is `&test`.

&test

Usage:

&test *testname* { *test_stmt_list* }

Arguments:

- *testname* is the name of this test.
- *test_stmt_list* is the list of statements that make up the test.

Description:

The `&test` statement defines a test block. There must be one test block for each test in a test table file.

Example:

See the complete example below.

Command

The commands are &process, &rc, &run, &set and &unset,

&process

Usage:

&process (*verb* : (*arguments*) [, *verb* : (*arguments*) …])

Arguments:

- *verb* is the name of the command to run. In the current version, the only verb supported is ivfnd.
- *arguments* are the arguments to the command.

Description:

The &process statement processes the results of the test run. It normally runs the IVFND program, which performs any filtering of the output and benchmark files and then compares the benchmark files. The return code from IVFND indicates the success or failure of the benchmark comparison. A return code of zero is success.

The use of the &process command is optional. However, if you choose to use the &process command you may not use it more than once per test block. If you do not use &process then you **must** use the &rc command in your test block.

Example:

```
&process( ivfnd : (-filter &infile(prs:logfilter) -result &resfile(log) ),
          ivfnd : (-filter &infile(prs:lstfilter) –result &resfile(lst) ) )
```

&rc

Usage:

&rc (*value*)

Arguments:

value is the expected value of the return code for this test.

Description:

The &rc statement tests the return code from the command specified in the &run statement. If the return code does not match the specified value, the test is marked as a failing test.

The use of the &rc command is optional. However, if you choose to use the &rc command you may not use it more than once per test block. If you do not use &rc then you **must** use the &process command in your test block.

Example:

```
&rc( 0 )
```

&run

Usage:

```
&run ( verb :  (arguments) )
```

Arguments:

- *verb* is the name of the program to run. In the initial version, the only verb allowed is “sas”.
- *arguments* is the list of options and arguments to the program being run.

Description:

The &run statement runs the actual test command. There must be **one** &run command in each test block. The &run must be followed by &process and/or &rc.

Example:

```
&run( sas : (-sysin &infile() -ls 78) )
```

This command runs the SAS system. The &infile statement (see below) passes an input file with the same base name as the test name. The other options are passed through unmodified.

&set

Usage:

```
&set ( name :  value )
```

Arguments:

- *name* is the name of the variable you are defining consisting only of alphanumeric characters.
- *value* is the any sequence of tokens. If the sequence contains parentheses, they must be balanced or the unbalanced parenthesis needs to be escaped (see example).

Description:

This command allows you define variables and assign values to them. The names and values you define using &set are global and remain in effect for the remainder of the current.

You may change the value of a variable by assigning a new value later using another &set statement.

Once you define a variable, you can use it anywhere in the table (as long as the &set statement precedes its use). To use the variable you created, use @name to insert the value of the variable. If you use a variable that has not been defined, an empty string is used for the value.

All parentheses inside the variable value must be balanced. If you require unbalanced parentheses, use the ‘\’ escape character to escape the parenthesis.

The &set statement may not be used inside of a block.

The testname and tablename variables are predefined for you and are set to the name of the test (or table) at the time the variable is expanded.

If the @ sign is needed in the value of the variable but should not be treated as a variable expansion, then escape the @ sign. See the examples below for a sample use of this.

Example:

```
&set( sasopts : -ls 78 -ps 60 )
```

This command sets the `sasopts` variable to "`-ls 78 –ps 60`".

```
&set( infile : -sysin &infile( sas : @testname : base) )
```

This command sets the `infile` variable to "`-sysin &infile( sas : @testname : base)`".

```
&set( infile1 : -sysin &infile\( sas : )
&set( infile2 : : base\) )
&set( run : sas : ( @infile1 @testname @infile2 \@noexpand) )
```

This command sets the `infile1` variable to `-sysin &infile( sas :`, `infile2` to `: base)` and `run` to `sas : ( @infile1 @testname @infile2 @noexpand)`. Assuming a test name of `test1`, the `run variable will expand to sas : ( -sysin &infile( sas : test1 : base)`.

&unset

Usage:

`&unset` (*`name`*)

Arguments:

- *`name`* is the name of a variable that may, or may not, have been set using the `&set` command. Note that the @ is **not** included when specifying the variable name, as that would cause the variable to be replaced with its value before the `unset` processing occurs.

Description:

This command allows you to guarantee that a variable is no longer set and if used, will expand to nothing.

Example:

```
&unset( testopts )
```

This command clears the `testopts` variable even if it had not previously been defined.

Macro

The macros are `&infile`, `&outfile`, and `&resfile`.

&infile

Usage:

`&infile(` *`ext [: name [: component]]`* **`)`** or
`&infile(`*`[ext]: name [: component]`***`)`** or
`&infile(`*`[ ext ] : [name] : component`***`)`**

Arguments:

- *`ext`* is the file extension. If it is omitted, the default is "sas".
- *`name`* is the file name. If it is omitted, the name of the test is used as the file name.
- *`component`* is the component area where the file is installed. If it is omitted, the component area where the table file was found is used.

Description:

The `&infile` statement defines an input file. It is normally used as part of an option to a command in the `&run` statement. Note that it is allowed to omit all of the arguments to `&infile`.

Example:

`&infile( sas : foobar : base )`
Specifies the file named `foobar.sas` from the base testware area.

`&infile()`

Specifies the file `testname.sas` (where testname is the name of the current test) from the testware area of the component where the `.tab` file is located.

&outfile

Usage:

`&outfile(` *`ext`* `[:` *`name`* `])`

Arguments:

- *`ext`* is the file extension. There is no default for the extension. It must be specified.
- *`name`* is the file name. If it is omitted, the name of the test is used as the file name.

Description:

The `&outfile` statement defines an output file. It is normally used as part of an option to a command in the `&run` statement.

`&outfile( lst : foo )`

Specifies the file named `foo.lst`.

`&outfile(log)`

Specifies the file `testname.log`.

&resfile

Usage:

&resfile(*ext [: name [: component]]***)** or
&resfile(*ext: name [: component]***)**

Arguments:

- *ext* is the file extension. There is no default for the extension. It must be specified.
- *name* is the file name. If it is omitted, the name of the test is used as the file name.
- *component* is the component area where the corresponding benchmark file is installed. If it is omitted, the component area where the table file was found is used.

Description:

The &resfile statement defines a test result file and a corresponding benchmark. It is normally used as part of an option to the IVFND command in the &process statement. It actually specifies a pair of files. One file is the output file from the test. It is assumed to be written to the current working directory where the test was run. The other file is the shipped benchmark file. It is in the specified location. The base names and extensions of the files must be the same.

Example:

```
&resfile( log : foobar : test : base )
```

Specifies the file named foobar.log from the base testware area.

```
&resfile( lst )
```

Specifies the file testname.lst (where testname is the name of the current test) from the testware area of the component where the .tab file is located.

Examples

```
&set ( sasopts : -sysin &infile -log &outfile( log : barfoo ) )
&test foobar
{
    &run( sas : (@sasopts) )
    &process( ivfnd : ( -filter &infile( prs : logfilter ) –result &resfile( log, barfoo ) ),
              ivfnd : ( -result &resfile( lst ) ) )
    &rc( 0 )
}
```

This test case runs the SAS system with an input file named foobar.sas. The file is in the same component area as the test table file. The test is expected to produce two output files, fooboo.log and foobar.lst. The log file and the corresponding benchmark are filtered using ped with the filter script named logfilter.prs (also in the testware area of the current component) before being compared. The lst file is not filtered before being compared. The SAS command is expected to exit with a return code of zero.

SAS Operational Qualification Tool Command Line

The SAS OQ command line arguments are defined as follows:

`-tables` *`table_spec [table_spec …]`*

The `–tables` option specifies which test tables to run. A `table_spec` has the form *`name:component`*. This is very similar to the specifications in the `&infile` and `&resfile` statements, except that the extension is always assumed to be `.tab`. At least one `table_spec` is required.

`-help`

The `–help` option requests the tool to print its usage.

`-verbose | -quiet`

These two arguments dictate the amount of STDOUT output the SAS OQ writes during execution. The `–verbose` option tells the SAS OQ to print extra output to STDOUT when executing. This is useful when setting up new tests. The `–quiet` option tells the SAS OQ to only report errors. This is useful when scripting SAS OQ commands.

`-sasroot <path>`

This argument tells the SAS OQ where to locate The SAS System installation. By default, the tool will attempt to locate the SASROOT path itself. If it is unable to, for some reason, this argument can be used to explicitly set that path. This can also be used to test different SAS System installations on the same machine.

`-testware <path>`

This argument specifies the location of the testware. By default `SASROOT/sastest` is used.

`-outdir <path>`

This argument specifies the location where all output files should be written. The default is to use the current directory where the tool execution took place.

Perl Stream Editor Documentation

Perl Stream Editor (`ped`) is designed to be an alternative to using `sed` (the original UNIX SysV Stream Editor). Although based on `sed`, the filter expressions that `ped` recognizes are different than those recognized by `sed`. Some filter expressions that are valid for `sed` are still valid for `ped`; but some are not! `ped` is not intended to translate existing `sed` filter files.

The biggest benefit to using `ped` is that you now have the power of Perl Regular Expressions at your disposal rather than the useful, but limited set of `sed` regular expressions. Another benefit to using `ped` is that it allows a syntax for inclusion of other filter files, similar to the inclusion process in many other languages.

`ped` also defines an extension to the Perl-provided set of regular expression commands. This command is called Substringing Substitution. It allows you to write a regular expression that identifies a substring of the input stream and then apply a Substitute regular expression command on only that substring. This allows you to be very specific about what you are filtering and it makes the actual search pattern regular expressions much, much less complicated. Briefly, the command syntax is: `S/<substring-RE>/<substring-Options>/<search-RE>/<replacement-RE>/<substitute-Options>`

The `substring-Options` are the same set of options available to the Perl-provided Substitute command. The Substringing Substitution command is implemented as a pair of Substitute commands.

What Can I Do with Perl REs That I Can't Do with sed REs?

Perl REs have the ability to match patterns over any number of newline characters. It has the ability to match even deeply nested patterns, this is very useful for markup languages like HTML, XML, and SGML. It has a much richer set of meta-sequences (white-space, word-characters, octal- and hex- characters, digits, etc.) You can capture matched sub-strings in memory buffers and substitute them back into the output string. You can do "look-aheads" and "look-behinds". You can easily express quantifiers and define new character classes. It has an extended syntax that allows you to add human-readable comments directly into complicated RE's thus allowing other folks to understand them and maintain them.

Also, `ped` has no limit on the number of filter expressions that it can accept. `sed` limits you to 100 expressions.

Filter File Syntax

- Lines that begin with a pound-sign (#) are ignored. This is the `ped` comment, it is the same for `sed`.
- Lines that contain nothing but whites pace are ignored.
- Lines that begin with `%include` are interpreted as include commands and not as REs. `ped` looks for a token following the `%include` ; it interprets that token as a file path. It attempts to read that file and process its contents as additional REs.

ped Command Line Syntax

```
ped [-e <'|"><filter RE><'|">] [-f <filter file path>] [-n] [<file path 1> [<file path n>]]
ped <'|"><filter RE><'|"> [<file path 1> [<file path n>]]
```

Substringing Substitute("S" command) Syntax

```
S/<substring-RE>/<substring-Options>/<search-RE>/<replacement-RE>/<substitute-Options>
```

Where substring-Options are defined to be the same set of options available to the Perl regular expression Substitute("s") command. The substring-RE and the search-RE are required. The substring-Options, replacement-RE, and substitute-Options are not required.

The "S"-command works by applying the substring-RE to the input stream; when a match is found, a common "s" command is applied to only the matched portion at portion of the input stream.

Ped Filter File Examples

mylogfilter.prs:

```
#--------------------------------------------------------------------
#  Purpose: To filter out the common, non-component-specific stuff
#           from SAS .log files.
#   Filter: ped
#--------------------------------------------------------------------
# Get rid of the page headers
s/^\f*\d{1,6}\s+The SAS System\s+$//gmo
# Get rid of the copyright notice.
s/^\s*note:\s*copyright\(c\).*$//imo
# Get rid of the "proprietary/licensed" note.
s/^\s*note:\s*sas\s*\(r\)\s*proprietary.+?\n\s*licensed.+?site.+?$//imos
# Get rid of the Institute address note.
s/^\s*note:\s*sas\s+institute\s+inc\.,\s+sas\s+campus\s+drive,\s+cary,\s+nc.+?$//imo
# Get rid of the "SAS used" message.
s/^\s*note:\s+the\s+sas\s+system\s+used:\s*\n\s+real\s+time\s+\d+\.\d\d\s+seconds\s*\n\s+cpu\s+time
\s+\d+\.\d\d\s+seconds\s+$//imos
```

Issues

Since `ped` reads in the entire input stream into memory before it begins to operate on it, the size of the file that `ped` can filter is limited to the amount of available memory. Fortunately, memory is large today and no file that is too large to filter has yet been reported. `sed` only reads a few lines at a time, so it can process files of virtually any size.

IVFND Command Line

IVFND (Install Validation Filter 'N' Diff) performs filtering and differencing operations for the SAS OQ. It is normally invoked from the `&process` statements in the test tables.

`Arguments:`

`-filter` *`ext:name:component`*

The `–filter` option specifies the name and location of the filter input file. If this option is present, IVFND executes PED using the specified filter script to filter both the test output file and the corresponding benchmark file before comparing them. Note that all of the fields must be present.

`-result` *`ext:name:component`*

The `–result` option specifies the name and location of the result file and of the corresponding benchmark file. The result and benchmark are filtered (if the `–filter` option is present) and then compared using IVDIFF.

Return codes:

Zero - All input files were found, filtering (if any) was successful, and the filtered files compared cleanly.

Non-zero - Something went wrong. Messages are written to `stdout/stderr`. If the files did not compare cleanly, difference files are left in the current working directory so the user can see exactly what failed. If any of the IVFND commands returns a non-zero return code, the test is marked as failing.

support.sas.com

SAS is the world leader in providing software and services that enable customers to transform data from all areas of their business into intelligence. SAS solutions help organizations make better, more informed decisions and maximize customer, supplier, and organizational relationships. For more than 30 years, SAS has been giving customers around the world The Power to Know®. Visit us at **www.sas.com**.