SDLTRACE FOR COBOL

Reference

Trace and Logging Facility for COBOL

1. Introduction

SDLTRACE for COBOL is an application system to debug, verify and log COBOL programs on the IBM® mainframe operating systems MVS® and z/OS®. This Reference describes all the features available and is intended to be used together with the User Guide, where a number of examples show how best to use and operate the system. It is therefore recommended to at least look at the samples provided in the User Guide to get to know SDLTRACE.

The main interface between the user and SDLTRACE FOR COBOL is an ISPF panel that is used to specify all parameters necessary for operation of the system. It serves to adapt the user's application program to the environment of SDLTRACE for the desired functions. It is the only panel to be completed by the user in order to set up a program for specific trace or logging operations to be performed. No other preparations such as allocation of datasets, changes to JCL, special provision for certain environments or any other activities are necessary. Once a program has been adapted using the specifications of this panel it can be compiled, installed in any environment from simple batch, to CICS®, IMS®, DB/2®, WLM®, etc., and then run, producing the requested results.

The SDLTRACE panel is stored in the distribution library under the name ATRACE (please see the User Guide) and will display the following (or similar) information when called up (make sure that the PF key display is turned off beforehand by entering **pfshow off** on the ISPF primary option menu):

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                             Screen 1 of 1
Run Jcl or eXec now or Delete or display Next/Previous screen: _ (J X D N P)
                                                             (or: I/R/E)
Input dataset        user-id.SDLTRACE.DEMO.COBOL_________________
Output dataset       user-id SDLTRACE.DEMO.COBOL_________________
Input member         SAMPLB01
Output member        SAMPLBX1        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     T (T/L)
Application-ID       SDLAPPL1        Trace PERFORM      N (Y/N)
JOB-ID check         *_______        Trace PERFORM end  N (Y/N)
DSN alloc (tracks)   100_ (1-9999)   Trace labels       N (Y/N)
DSN time (minutes)   5___ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     Y (Y/N)         Include string #1  ______________________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     N (Y/N)         Include string #3  ______________________
Enable CICS test     N (Y/N)         Exclude string #1  ______________________
Local time / GMT     L (L/G)         Exclude string #2  ______________________
Enable timing        N (Y/N)         Exclude string #3  ______________________
Timing threshold ____0 (0-32767 ms)  Pgm1 ________ Pgm2 ________ Pgm3 ________

F3 = Quit                                              ENTER = Process input
```

There are two distinct areas for input on this panel: the top line beginning with **`Run Jcl`** which is used to specify an action to be performed, and all other entry fields on the screen which are used to specify data.

The two entries "**j**" and "**x**" specify execution of the panel, "**j**" for batch job mode and "**x**" for TSO online execution.There may be up to nine copies of the panel with different parameters that are stored in the user's ISPF profile dataset. The letters "**n**" and "**p**" are used to switch between them. A panel that is not needed anymore can be deleted by specifying "**d**". Lower case entries are automatically translated to upper case. The additional values "**i**", "**r**" and "**e**" are simple shortcuts for specifying them first in the **`Insert/Remove/Edit`** field and then "**x**" in the action entry.

2. Input dataset, Output dataset, member

The SDLTRACE system for tracing and logging COBOL programs applies certain changes to the user's source program in order to generate the requested information. In this section of the ATRACE panel the user specifies the Input dataset and the member name of the module to be used as source for the action defined in the "Insert/Remove/Edit" field. Likewise "Output dataset" and "Output member" denote the target where the modified program module is to be stored.

The Input dataset may be the same as the Output dataset, and the Input member name may be identical to the Output member name, in which case the original is replaced with the modified program. For the action "Edit" this is required, in all other cases the datasets and member names may be different.

There is a special case for member names: If '*' is specified for the Input member and the Output member then all programs in the Input library are to be modified at once, one after the other, and to be placed in the Output dataset. In this case the name of the Output dataset must be different from the Input dataset.

3. Insert/Remove/Edit

There are two distinct actions which may be specified as operation when the panel is being executed:

“Insert” and “Remove”

“Insert” means that the user’s source program is to be modified to include all necessary statements to activate the requested trace or logging operations.

“Remove” is just the opposite: It serves to delete all previously inserted statements.

In both cases the modified source code, either with SDLTRACE statements (action “Insert”) or without any SDLTRACE statements (action “Remove”), is displayed in the ISPF edit panel for further editing or compilation.

The special action “Edit” is a combination of the two actions above. It requires that the input and output datasets and members are identical and starts with a “Remove” action as first step. As a result the original program without any SDLTRACE modifications is displayed for editing in the ISPF edit panel. When “PF3” is hit, the “Insert” action is performed as second step to equip the program with all modifications that were specified in the ATRACE panel in the beginning. Again, the program is displayed in the ISPF edit panel for any further editing or compilation.

4. DSN qualifier

All data that is being generated by trace or logging calls to SDLTRACE is stored in dynamically allocated datasets. The names of these datasets are generated from three sources: Specifications by the user, information from the system on which the program is being executed and certain fixed markers used for identification. The general format of the trace/log datasets is.

DSN qualifier.Application-ID.jobname.**zz**.**D**yymmdd.**T**hhmm**x**

where “**x**” may be either “**A**”, “**B**”, “**C**”, which denote trace datasets or “**L**” which is used exclusively by the logging function to identify datasets with log data contents. The DSN qualifier specifies the first level and may be any valid dataset name for which the job or monitor, under which the program is being executed, does have “Write access” authorization. In case that this authorization is missing, it will not be possible to allocate the trace dataset and an error message will be issued. No trace or log data will be produced in this case; the program, however, will be executed as if no trace/log code were present.

The DSN qualifier may consist of more than a single name; thus levels such as “ABC.LOG” or “XYZ.TEST” are perfectly legal. In many cases, especially in batch test environments under TSO, the current user-id is taken as DSN qualifier. If the module under test is to be executed under CICS however, it will be necessary to change to a name for which the CICS monitor has allocation privileges.

zz are two characters which represent the job-id of the job or monitor being executed at the time of allocation. These two characters may be ‘A’ through ‘Z’ for the first one and ‘A’ through ‘Z’ and ‘0’ through ‘9’ for the second one. The Job-Id is used to compute this two-character hashcode which then becomes part of the log or trace dataset name. This is necessary in order to guarantee unique names in environments where there are multiple jobs with the same name running concurrently, as for example in WLM.

There may be instances where a large number of individual jobs is generated automatically, which run only very briefly, one after another, under the same jobname, but each with its own job-id. Each job will create its own log dataset, creating possibly many thousand datasets within a very short period of time. As long as they run one after the other there really is no need to distinguish them by their job-id; their log data might as well be written into one log dataset.

For this situation, the following JCL statement is provided:

```
//SDLJOBNM DD DUMMY
```

If this statement is encountered in the job step stream, the hashcode is computed from the jobname, and not from the job-id, so that the log or trace dataset name will be identical for all jobs generated by this application.

5. Application-ID

The Application-ID is the second major part of the dynamically allocated trace or log datasets. Any valid DSN may be specified, either a single name or a combination such as “APP.TST1”. If an application-ID is not specified (entry field is left blank) then the name of the program itself is taken and used to construct the trace/log dataset name.

The Application-ID can be used to direct trace/log output to a single dataset even though the individual records are generated in separate programs. Within a job all programs with trace/log code write their generated data to the file “DSN qualifier.Application-ID.jobname.xx.Dyymmdd.Thhmmx”. If the Application-ID (and of course the DSN qualifier) are identical in the programs being called within a job then the trace/log data will combined into that dataset.

6. JOB-ID check

The JOB-ID check can be used to limit the generation of trace/log data to only specific jobs or monitors. Usually this field is set to “*” (or left blank) which means that trace/log data is always generated. When a job name is specified then the data is generated only when the program is executed within a job whose name corresponds to the specified name.

The job name may also be specified only partially qualified, that is the first part of the name followed by “*” as for example in: TEST*. Only for jobs whose name starts with ”TEST” will there be any generation of trace/log data. Likewise for online monitors such as “CICS*”. In this case only when the program with trace/log code is executed in a monitor whose name starts with ”CICS” will there be any generation of relevant data.

A special case is the string “JOB-LIST”. This specifies that the decision if trace/log data is to be produced is based on information contained in a list which is linked together with SDLTRACE at installation time. This list may be changed at any time and SDLTRACE relinked with the changed information.
The distribution library contains a job called “SDLJOBLG” which is used to do that.

SDLJOBLG is a small Assembly program that contains just a list of names, built by using the Macros LOG and NOLOG. This list specifies the job or monitor names for which tracing or logging is to be done, and optionally those names for which tracing or logging is to be suppressed. In addition, a high level qualifier can optionally be specified as second parameter. If it is, it will be used instead of the name specified in the “DSN qualifier” field. (Special high qualifier override for specific job names).

For more information on this feature and how to use it please refer to the comments in SDLJOBLG.

7. DSN alloc

The space parameter `DSN alloc (tracks)` specifies the number of tracks between 1 and 9999 to be allocated when a trace or log dataset is required. The amount depends on the particular application and the expected size of the trace/log data that will be generated. Because often that is not known, the system extends the initial allocation when needed. There are major differences between tracing and logging as far as extension of datasets is concerned. For tracing the procedure is as follows:

A second dataset will be allocated when the first dataset has reached the number of tracks specified. This second dataset has the same number of tracks and the same name as the first one, except that the last letter is ‘B’ instead of ‘A’. Trace recording continues on dataset ‘B’ until that too has reached its limit, at which point a third dataset is allocated with the same dataset name and the last letter ‘C’. When the limit is reached on ‘C’ then the trace continues recording again on ‘B’, where the previously recorded trace data is overwritten, then ‘C’ again, overwriting the previous data too, and so on, alternating between ‘B’ and ‘C’ indefinitively. Thus there is trace information from the start of the program in dataset ‘A’, and the last statements executed can be found either in ‘B’ or in ‘C’.

For logging there is the requirement that no data is being lost, so that nothing can be overwritten. Therefore a new dataset will be allocated as soon as the current one is full. The name is the same except for the last DSN level **T**hhmm**L** which reflects the actual time of allocation. There is no limit to the number of datasets that might be allocated because that depends entirely on the amount of log records being generated by the application.

8. DSN-time

The parameter ”`DSN time (minutes)`” is used to limit the active time of a trace or log dataset. This feature is mostly used with logging; for tracing it is still available but not really needed.

A time between 1 and 1440 minutes can be specified with the restriction that the value must be divisible into 1440 without remainder, so that there is a whole number of periods during one day (1440 minutes). Whenever an interval is complete, the currently active dataset is closed and logging (or tracing) continues with recording on a new dataset. A value of zero indicates that no new dataset is to be allocated during an entire day and corresponds to a specification of 1440. The recommended value for logging is `60` so that there is at least one new dataset every hour.

Irrespective of the setting for `DSN time (minutes)` a new dataset will always be allocated on midnight, since then the day changes and therefore the date part of the DSN must be changed too. The reason for this is the rule that all log data from a particular day is only recorded in datasets whose names indicate exactly that day.

The process of creating new datasets periodically and switching to a new name on midnight can go on indefinitively, and especially in long running jobs or online monitors which run for weeks or months without interruption, this feature is used extensively by the logging function of SDLTRACE.

9. Trace/Log mode

This parameter is used to define the mode in which SDLTRACE is to operate: “T” denotes tracing and “L” specifies logging. Most of the features of SDLTRACE apply to both, tracing and logging. The differences between the two modes are in the following areas:

> In trace mode the last character of the trace/log dataset name is “A”, “B” or “C”, whereas in log mode it is always “L”.
>
> In log mode all variables of length 80 (or an exact multiple of 80) are not formatted according to their type, and the variable names are not displayed at all; instead the variable content is moved to the log record as is without any modification (except the insertion of execution times, if desired).
>
> In trace mode the execution times are displayed in separate records whereas in log mode the times are inserted into the log record (if the relevant area is blank).
>
> In log mode the name of the calling program is inserted into the log record (and optionally the caller of the caller, etc.). In trace mode the name of the program being traced is displayed rather than the caller.
>
> In trace mode the trace dataset is closed upon exit from the module being traced and re-opened when the module is entered again. In log mode the log dataset is not closed when the module being logged is returning control to its caller. Closing of log datasets is performed depending on the setting of the space and time parameters `DSN alloc` and `DSN time` on the ATRACE panel.

It is important that in addition to setting “`Trace/Log mode`“ to “L” the length of the variables supplied to the system for logging is exactly 80 (or an exact multiple of 80) bytes long. Otherwise they will be handled as in trace mode and formatted accordingly. In this way it is possible to combine tracing and logging within one program, which sometimes might be useful. Thus the logging mode is essentially a special case of tracing, with features tailored to the requirements of an efficient logging system.

10. Trace PERFORM

With the "`Trace PERFORM`" parameter set to "Y" the system will generate a trace record denoting the execution of every "PERFORM paragraph" statement in the program. All other PERFORM statements are not traced. In log mode this parameter should be set to "N".

11. Trace PERFORM end

The "`Trace PERFORM end`" parameter specifies if special code should be generated to trace the end of a "PERFORM paragraph" or section. In log mode this parameter should be set to "N".

12. Trace labels

If "`Trace labels`" is set to "Y" then the system will generate a trace record whenever a label is being encountered during execution of the program. This can be used independently of the `Trace PERFORM` and `Trace PERFORM end` parameters and shows the flow through a program independently of paragraphs and sections. In log mode this parameter should be set to "N".

13. Trace variables

The parameter "`Trace variables`" specifies if variables are to be traced or logged. It is almost always set to "Y", since it is the change of the contents of variables during execution of a program that is of most interest. Only in cases where the flow through a program is to be documented should the parameter be set to "N" and one or more of the three parameters above to "Y".

With "`Trace variables`" set to "Y" most values assigned in "MOVE" or "SET" statements are recorded in the trace/log file, unless there are entries in the "Include" and "Exclude" fields below which can be used to restrict tracing or logging to certain names. Only MOVE statements which have a single variable as target are being traced. When a list of names is the target of a MOVE statement then no trace data is being generated. This, however, can easily be changed by inserting a MOVE with just a single variable as target.

In addition to tracing variables of MOVE and SET instructions, SDLTRACE also records the results of SQL FETCH statements. As usual these variable names are preceded by ":" to document that their values were set from tables in a database. When specified in "Include" or "Exclude" statements below the leading ":" should however be omitted.

As already mentioned, the variables to be traced or logged can be restricted to those specified in the parameters "`Include string …`" and "`Exclude string …`" described further down.

14. Count duplicates

With the parameter “`Count duplicates`“ it is possible to control how identical records are to be treated.

When “`Count duplicates`“ is set to “N” then no check is made to determine if succeeding records being recorded in the trace/log file are identical or not. They are all written to the dataset, and the duplication factor is set to 1.

With “`Count duplicates`” set to “Y” the system checks the records to be written for identical content, and if that is the case then the second identical record and succeeding ones are not written until a record with different content is encountered. The last identical record is then written together with the updated duplication count stored in the record.

In trace mode this parameter should be set to “Y” in order to take advantage of the possible reduction in output, especially with large variables containing many blanks.

In log mode the check for duplicate records is often not necessary and sometimes it is even not desired. However, if large records of varying size are being generated then it may be useful to set the parameter to “Y”.

15. Console messages

There are a number of messages being displayed on the operating systems job log console during execution. The most informative messages for the user are the names of the datasets being allocated since they will contain the trace or log data that was generated. The display of these names is controlled by the setting of the parameter “`Console messages`”, which in most cases is set to “Y”.

However, in production environments where the log datasets are collected regularly and sent to other systems for consolidation it may not be necessary to display all allocated dataset names. Just set “`Console messages`“ to “N” to achieve this. One should be aware though that sometimes there are warning or error messages which might then also not be listed.

The SDLTRACE starting message containing information about the current version and licensing conditions cannot be suppressed; it as always displayed.

16. Save RETURN-CODE

The parameter “`Save RETURN-CODE`“ is used to control the treatment of passage of the contents of Register 15 within the sequence of application calls. The COBOL special variable RETURN-CODE (Register 15) is sometimes being used by applications to pass information between different modules, and any interference by other programs might change the desired results.

Since the invocation of SDLTRACE is an external call which always changes Register 15 there may be side effects in applications that rely on passing of the COBOL RETURN-CODE (Register 15). Such effects are avoided by setting “`Save RETURN-CODE`“ to “Y”', which will preserve the contents of the RETURN-CODE variable. It is the recommended setting and should almost always be used.

In special cases where it is necessary to get the result of a trace call by checking Register 15 upon return from the trace/log call, the parameter “`Save RETURN-CODE`“ can be set to “N”. This is recommended only when precautions have been taken against the possible side effects mentioned above.

17. Enable CICS test

Note: This parameter applies to log mode only (“`Trace/Log mode`“ set to “L”). In trace mode this parameter has no effect.

The parameter “`Enable CICS test`“ is used when testing logging applications for online environments such as CICS, and applies to all online monitors, not only CICS. It concerns the disposition of the log dataset after the program that calls the log routine is finished and the online transaction is closed.

The default for log mode is to keep the dataset open for further logging activity after a transaction is complete. Therefore the log dataset will remain in use by the environment. That means that the user designing and testing the code cannot look at the log results immediately. Rather the programmer has to wait until the dataset is freed either by waiting until the active time interval of the log dataset expires or the entire online environment is temporarily stopped and then started again. Please note that this applies to log mode only; in trace mode the dataset is always closed after exit from the trace call so that this situation does not arise.

When “`Enable CICS test`“ is set to “Y” the default of the trace mode is also applied to all logging calls so that the log dataset is available for browsing immediately after the transaction is executed. The impact on performance is very high, so that it is not recommended at all to use this setting in production. For testing however, where performance is of no concern, this setting is very convenient. In all other cases “`Enable CICS test`“ should be set to “N”.

Another method that can be used to achieve almost instant access to the contents of log datasets is the setting of the parameter ”`DSN time (minutes)`”. If this is set to 1 then a new log dataset will be allocated every minute so that results are available relatively quickly. Please note that it is not enough to just wait one minute; there must be some action taking place to force the allocation of a new dataset (and free the previous one) before the current dataset can be viewed by the user.

18. Local time / GMT

For each trace/log record a timestamp will be generated and stored together with the data in the trace/log dataset. This timestamp represents the actual time of the call and is obtained from the operating system.

Two different settings are possible:

“`Local time / GMT` “ set to “L” denoting local time, or “`Local time / GMT` “ set to “G” denoting universal time. Since SDLTRACE was developed at a time when “GMT” was still the preferred term for what is now “universal time” the parameter to use is “G” rather than “U” when universal time is desired.

In most instances this parameter will be set to “L”. However, since log data may be generated in different locations around the world and then transferred and consolidated in one site it may be useful to use the setting “G” so that the sequence of events taking place in remote systems may be analyzed. The system clocks should be synchronized beforehand of course to make sure that events really can be correlated to each other. A universal timestamp is marked in each log record with an asterisk “*” in column 107, immediately preceding the timestamp which starts in column 108.

19. Enable timing

As described in the previous parameter each log/trace record contains the date and time of its creation, either in local or in universal time (GMT). These times can be used to compute the difference between individual events and thus arrive at elapsed times for any interval of special interest.

The parameter “`Enable timing`“ specifies if these timestamps should be used by SDLTRACE to compute and then record execution times within the trace/log dataset. There is no additional information being gathered by the system (except perhaps CPU times, please see further down). All elapsed times could also later be computed manually by the user through analysis of the trace/log timestamps. Actually the parameter should have been called “`Enable timing analysis`“ to more accurately describe its action.

With “`Enable timing`“ set to “Y” the system determines the difference between consecutive (or specially defined) timestamps and records the results in the trace/log dataset. In trace mode additional records are being created and written, whereas in log mode the computed times are stored in the log records at position 70 through 80 if no user data is present (the area must be blank). The times being recorded are: CPU time at position 70 and elapsed time at position 76. In trace mode two separate records are created and written to the trace dataset, one for CPU time and one for elapsed time.

With “`Enable timing`“ set to “N” the system does not compute any time differences and no additional data is being recorded in the trace/log datasets.

In order to get the elapsed time between two specific places in the application program, additional calls to SDLTRACE have been provided in a special feature, described in the User Guide:

`'GET TRACE TIMESTAMP'` and `'SET TRACE TIMESTAMP'`.

For an explanation please look at the description of example program SAMPLH41 in the User Guide.

20. Timing threshold

When timing is enabled the recording of execution times can be restricted to values above a certain threshold by specifying a number between 1 and 32767. This limits the recording of the times to those values only which are above the specified threshold. The number denotes milliseconds and refers to the elapsed time value only. The CPU time is not considered. If the elapsed time is below the number specified then neither time will be recorded.

21. Include string #1, #2, #3

Usually all variables are selected for tracing. With this parameter it is possible to restrict tracing (and logging) to those variables only which contain the specified strings. Up to three different strings may be specified. All variables which do not contain any one of the specified strings will not be traced (or logged). The comparison to determine equivalence is not case-sensitive.

22. Exclude string #1, #2, #3

This parameter is used to exclude specific variables from tracing (or logging). When “Exclude string” values are specified then all those variables which contain any of the specified strings will not be traced (or logged). Up to three different strings may be specified. The comparison to determine equivalence is not case-sensitive.

23. Pgm1   Pgm2   Pgm3

When logging mode is active the standard log record includes the name of the program that called the program currently running. Sometimes it is useful to get the name of the program which called the current program, or even the name of the one which called the caller. Up to three program names may be specified, and if one of them is encountered in the call chain, it will be skipped and the next higher one in the chain will be taken, until a name is found which is neither “IGZ…..” (system module) nor Pgm1, Pgm2 or Pgm3.

SPECIAL NOTE: The determination of the name of the calling program is using the fact that the caller’s name is stored in the save-area. Alas, no more! With COBOL 6.1 this valuable information is not available anymore, at least not in the save-area the way it used to be. One may hope that this was an oversight that will be corrected in the future. (Remark Jan. 14, 2020, HL).

24. Disable logging through JCL

Sometimes it may be necessary to disable logging or tracing for a certain job or monitor quickly without changing any program. And although the SDLTRACE system has been designed such that no JCL-changes are required to **en**able logging once the actual logging code is in the programs, there is a feature that allows logging to be **dis**abled simply by inserting a JCL-statement into the job-step where logging is to be suspended. The statement is:

```
//SDLNOLOG DD DUMMY
```

Whenever this statement is found in the job-stream no log-data will be generated and no log-file will be created. The logging instructions within the programs are just ignored.

25. Close log-files after each call through JCL

In some very special environments the system may not tolerate the fact that the logging files stay open from one call to the next, the feature that actually allows SDLTRACE to run in production without causing any performance problems. If there is an instance where an open file might be the cause of an error it may be necessary to disable that feature through JCL. The statement is:

```
//SDLCLOSE DD DUMMY
```

Whenever this statement is found in the job-stream the current log-file will be closed before returning to the caller. The impact on performance is substantial in this case, and it is therefore not recommended to be used in large scale production. However, if the number of log-records generated is limited, then there is no reason why it couldn’t be used.

The statement is also very useful during development since it guarantees that every log-record is written to the file immediately after its generation rather than being kept in some buffer.