SDLTRACE FOR COBOL

User Guide

Introduction to Trace Facility for COBOL

For every COBOL programmer the DISPLAY statement is probably the easiest and most often used method to get information about the behavior of a program. In a sense SDLTRACE is an extension of that procedure: At relevant places in a program calls to SDLTRACE are inserted, and the trace module records the information provided together with additional information necessary for later analysis. The data is written to datasets which are automatically allocated by SDLTRACE and managed in such a way that recording can go on indefinitely. No changes to the runtime JCL is needed, and the environment can be anything from simple batch, to CICS, IMS, DB/2, WLM, etc.. For analysis, the recorded data can be merged and sorted, such that events in completely separated subsystems can be correlated to each other, since each event is tagged with the exact time it occurred. Thus even widely distributed application systems can be analyzed and possible sources of malfunction identified.

Chapter 1.   Tracing a simple COBOL program

There are six small programs provided in the COBOL library to show how to use the trace facility.
You may need to edit the job cards of the members in dataset “*user-id”.*SDLTRACE.DEMO.COBOL before submitting the jobs, for example to insert accounting information required by your installation.

In the following module (SAMPLB01 in library „user-id“.SDLTRACE.DEMO.COBOL) just a few MOVE statements are being executed. As stored in the library, the module also contains all necessary JCL to compile, link and run it. The COBOL code (without JCL) is as follows:

```
       ID DIVISION.

       PROGRAM-ID  SAMPLB01.

       DATA DIVISION.

       WORKING-STORAGE SECTION.
       01  CHARACTER-DATA            PIC  X(20)  VALUE SPACE.
       01  CHARACTER-DATA-LONG       PIC  X(256) VALUE SPACE.
       01  NUMERIC-DATA-UNSIGNED     PIC  9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-POS   PIC S9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-NEG   PIC S9(8)   VALUE ZERO.
       01  DECIMAL-DATA-UNSIGNED     PIC  9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-POS   PIC S9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-NEG   PIC S9(7)   COMP-3 VALUE ZERO.
       01  BINARY-DATA               PIC S9(9)   BINARY VALUE ZERO.

       PROCEDURE DIVISION.
           MOVE 'Hello, COBOL!' TO CHARACTER-DATA
           MOVE 'This is character data that extends over more than one
      -         'line and shows how data is displayed on several lines'
                                TO CHARACTER-DATA-LONG
           MOVE  123            TO NUMERIC-DATA-UNSIGNED
           MOVE  456            TO NUMERIC-DATA-SIGNED-POS
           MOVE -789            TO NUMERIC-DATA-SIGNED-NEG
           MOVE  123            TO DECIMAL-DATA-UNSIGNED
           MOVE  456            TO DECIMAL-DATA-SIGNED-POS
           MOVE -789            TO DECIMAL-DATA-SIGNED-NEG
           MOVE  123456789      TO BINARY-DATA
           GOBACK.

       END PROGRAM SAMPLB01.
```

The program does not do much: It executes the assignment statements and returns, no output is produced. To set up this module for execution with tracing just go into ISPF panel 3.4, display the members of „user-id".SDLTRACE.DEMO.CNTL, step down to member ATRACE and type "**ex**" (short for "exec") in front of it. The following panel will be displayed (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
```

This panel is used to specify parameters for the COBOL SDLTRACE pre-processor "SDLPREP", a REXX program that scans the module to be traced. There are two modes of operation for this pre-processor: batch and TSO. In batch mode a job is prepared that can be submitted for execution. In TSO mode the pre-processor is called directly from the panel. Selection between the two modes is made by specifying "**j**" or "**x**" respectively in the top right entry field on the panel. In the following examples we will always use "**x**" followed by the "`Enter`" key to submit the panel for execution.

There may be up to nine copies of the panel with different parameters that are stored in the user's ISPF profile dataset, and the letters "**n**" and "**p**" may be 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.

When the panel above is executed by entering "**x**" and hitting the "`Enter`" key, the following messages will be generated for the input values on the panel above, (provided that the values in your panel which you submit for execution are identical to those shown above):

```
SDLTRACE - Version 4.5.23      14 Mar 2015 08:11:38 user-id
SDLTRACE - Parameter file: „user-id".SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=TRACE
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - ------------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      SAMPLB01
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     SAMPLBX1
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      57
SDLTRACE - MOVE .... TO variables:            9
SDLTRACE - Number of lines inserted:        188
SDLTRACE - Number of lines with trace:      245
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

Identical output would be produced of course by selecting “**j**” and then submitting the generated job, which is executing the pre-processor in batch. In each case the parameters for execution of SDLPREP are transferred from the panel to “SDLPRCTL”, which is used by SDLPREP during execution.

The usual action for SDLPREP is INSERT, which means to prepare a module for tracing, whereas REMOVE is the exact opposite, namely removing all previously inserted trace statements to reproduce the original program again. The third action which may be specified is EDIT, a combination of REMOVE and INSERT. Since it is easier to work on a program when it is in its original form (without any trace statements present), it is advisable to remove the trace code before editing the program, and then insert the trace code again before the next compilation. This action is performed automatically with the EDIT function, which removes any trace statements before displaying the code for editing, and, after leaving the SPF-edit with PF3, reinserts the trace statements and again displays the program for further action. If JCL is part of the member (as in the sample modules) then the program may be submitted directly for translation and execution.

In the example above INSERT is specified to add code that will trace all variables whose values are changed by MOVE statements. The number of lines in the original is 57, which is the total number of lines in the input member SAMPLB01 including the JCL. There are 9 MOVE statements and a total of 188 lines are added to the code resulting in altogether 245 lines which are stored in output module SAMPLBX1.

If TSO mode (entry “**x**” on the panel) is used to invoke the pre-processor, then the changed source module will be displayed in the ISPF editor so that the inserted code can be reviewed and edited if necessary. The original COBOL program as changed during processing with the settings from the panel above now looks like this:

```
       ID DIVISION.

       PROGRAM-ID  SAMPLB01.

       DATA DIVISION.

       WORKING-STORAGE SECTION.
SDL#***------------------------------------------------------------***
SDL#Y  01  SDLTR-PARM  GLOBAL.
SDL#Y      05  SDLTR-VALIDCHK-A   PIC X(8)    VALUE  'SDLTRACE'.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VERSION      PIC X(8)    VALUE  'VER 4.5 '.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-DSN-HILEVEL  PIC X(8)    VALUE   SPACE.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-APPL-ID      PIC X(8)    VALUE   SPACE.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PGMNAME      PIC X(8)    VALUE   SPACE.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-JOBNAME      PIC X(8)    VALUE   SPACE.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TYPE         PIC X(5)    VALUE   SPACE.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TEXT         PIC X(50)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-START   PIC X(8)    VALUE  'PERFORM '.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-END     PIC X(8)    VALUE  '--END-- '.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-LABEL        PIC X(8)    VALUE  '------- '.
SDL#Y      05                     PIC X(1)    VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PRI-TRKS     PIC 9(4)    VALUE 250   BINARY.
SDL#Y      05  SDLTR-SEC-TRKS     PIC 9(4)    VALUE 250   BINARY.
SDL#Y      05  SDLTR-VAR-LENGTH   PIC 9(4)    VALUE 31    BINARY.
```

```
SDL#Y      05  SDLTR-FLD-LENGTH   PIC 9(4)   VALUE 13   BINARY.
SDL#Y      05  SDLTR-THRESHOLD    PIC 9(4)   VALUE 0    BINARY.
SDL#Y      05  SDLTR-NEWTIM       PIC 9(4)   VALUE 1440 BINARY.
SDL#Y      05  SDLTR-RETN-CODE    PIC 9(2)   VALUE   0.
SDL#Y      05  SDLTR-RETN-CBIN    PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TRACE        PIC 9(1)   VALUE   1.
SDL#Y        88 SDLTR-TRACE-ON               VALUE   1.
SDL#Y        88 SDLTR-TRACE-OFF              VALUE   0.
SDL#Y      05  SDLTR-CALLER       PIC X(1)   VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-ASM             VALUE   'A'.
SDL#Y        88 SDLTR-CALLER-COBOL           VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-C               VALUE   'C'.
SDL#Y      05  SDLTR-VAR-TYPE     PIC X(1)   VALUE   SPACE.
SDL#Y        88 SDLTR-VAR-TYPE-ALL           VALUE   ' '.
SDL#Y        88 SDLTR-VAR-TYPE-BIN           VALUE   'B'.
SDL#Y        88 SDLTR-VAR-TYPE-CHR           VALUE   'C'.
SDL#Y        88 SDLTR-VAR-TYPE-DEC           VALUE   'D'.
SDL#Y        88 SDLTR-VAR-TYPE-HEX           VALUE   'X'.
SDL#Y      05  SDLTR-TIMESTAMP    PIC X(1)   VALUE   'L'.
SDL#Y        88 SDLTR-TMSTP-GMT              VALUE   'G'.
SDL#Y        88 SDLTR-TMSTP-LOC              VALUE   'L'.
SDL#Y      05  SDLTR-WRITE-IMM    PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-WRITE-IMM-ON           VALUE    1.
SDL#Y        88 SDLTR-WRITE-IMM-OFF          VALUE    0.
SDL#Y      05  SDLTR-CONS-MSG-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-CONS-MSG-OFF           VALUE    1.
SDL#Y        88 SDLTR-CONS-MSG-ON            VALUE    0.
SDL#Y      05  SDLTR-TIMING-IND   PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TIMING-ON              VALUE    1.
SDL#Y        88 SDLTR-TIMING-OFF             VALUE    0.
SDL#Y      05  SDLTR-TRACECTL     PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TRACECTL-ON            VALUE    1.
SDL#Y        88 SDLTR-TRACECTL-OFF           VALUE    0.
SDL#Y      05  SDLTR-LOG-MODE-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-LOG-MODE-ON            VALUE    1.
SDL#Y        88 SDLTR-LOG-MODE-OFF           VALUE    0.
SDL#Y      05  SDLTR-DUPLICAT-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-DUPLICAT-ON            VALUE    1.
SDL#Y        88 SDLTR-DUPLICAT-OFF           VALUE    0.
SDL#Y      05  SDLTR-SKIP-NAME1   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME2   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME3   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-RESERVED     PIC X(18)  VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SYSTEM-AREA  PIC X(1800) VALUE  LOW-VALUE.
SDL#Y      05  SDLTR-VALIDCHK-Z   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y
SDL#Y  01  SDLTR-WORK  GLOBAL.
SDL#Y      05  SDLTR-LENGTH       PIC S9(9) BINARY.
SDL#Y      05  SDLTR-SAVERC       PIC S9(4) BINARY.
SDL#Y      05  SDLTR-INDEX1       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX2       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX1-NUM   PIC 9(8).
SDL#Y      05  SDLTR-INDEX2-NUM   PIC 9(8).
SDL#Y      05  SDLTR-SET-TRUE     PIC X(4)  VALUE 'TRUE'.
SDL#Y      05  SDLTR-GEN-DATE     PIC X(11) VALUE '14 Mar 2015'.
SDL#Y      05  SDLTRACE           PIC X(8)  VALUE 'SDLTRACE'.
SDL#***---------------------------------------------------------------***
       01  CHARACTER-DATA            PIC  X(20)  VALUE SPACE.
       01  CHARACTER-DATA-LONG       PIC  X(256) VALUE SPACE.
       01  NUMERIC-DATA-UNSIGNED     PIC   9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-POS   PIC S9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-NEG   PIC S9(8)   VALUE ZERO.
       01  DECIMAL-DATA-UNSIGNED     PIC  9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-POS   PIC S9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-NEG   PIC S9(7)   COMP-3 VALUE ZERO.
       01  BINARY-DATA               PIC S9(9)   BINARY VALUE ZERO.

       PROCEDURE DIVISION.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'SAMPLB01' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE 'SDLAPPL1' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 5          TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-OFF  TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-ON   TO TRUE
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL.
SDL#***------------------------------------------------------------***
           MOVE 'Hello, COBOL!' TO CHARACTER-DATA
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'CHARACTER-DATA' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         CHARACTER-DATA
SDL#H      BY CONTENT LENGTH OF CHARACTER-DATA
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE 'This is character data that extends over more than one
      -         'line and shows how data is displayed on several lines'
                                TO CHARACTER-DATA-LONG
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V2'       TO SDLTR-TYPE
SDL#F      MOVE 'CHARACTER-DATA-LONG' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         CHARACTER-DATA-LONG
SDL#H      BY CONTENT LENGTH OF CHARACTER-DATA-LONG
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE  123            TO NUMERIC-DATA-UNSIGNED
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V3'       TO SDLTR-TYPE
SDL#F      MOVE 'NUMERIC-DATA-UNSIGNED' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         NUMERIC-DATA-UNSIGNED
SDL#H      BY CONTENT LENGTH OF NUMERIC-DATA-UNSIGNED
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE  456            TO NUMERIC-DATA-SIGNED-POS
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V4'       TO SDLTR-TYPE
SDL#F      MOVE 'NUMERIC-DATA-SIGNED-POS' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         NUMERIC-DATA-SIGNED-POS
SDL#H      BY CONTENT LENGTH OF NUMERIC-DATA-SIGNED-POS
```

```
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE -789            TO NUMERIC-DATA-SIGNED-NEG
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V5'       TO SDLTR-TYPE
SDL#F      MOVE 'NUMERIC-DATA-SIGNED-NEG' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         NUMERIC-DATA-SIGNED-NEG
SDL#H      BY CONTENT LENGTH OF NUMERIC-DATA-SIGNED-NEG
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE  123            TO DECIMAL-DATA-UNSIGNED
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V6'       TO SDLTR-TYPE
SDL#F      MOVE 'DECIMAL-DATA-UNSIGNED' TO SDLTR-TEXT
SDL#U      SET   SDLTR-VAR-TYPE-DEC TO TRUE
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         DECIMAL-DATA-UNSIGNED
SDL#H      BY CONTENT LENGTH OF DECIMAL-DATA-UNSIGNED
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE  456            TO DECIMAL-DATA-SIGNED-POS
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V7'       TO SDLTR-TYPE
SDL#F      MOVE 'DECIMAL-DATA-SIGNED-POS' TO SDLTR-TEXT
SDL#U      SET   SDLTR-VAR-TYPE-DEC TO TRUE
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         DECIMAL-DATA-SIGNED-POS
SDL#H      BY CONTENT LENGTH OF DECIMAL-DATA-SIGNED-POS
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE -789            TO DECIMAL-DATA-SIGNED-NEG
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V8'       TO SDLTR-TYPE
SDL#F      MOVE 'DECIMAL-DATA-SIGNED-NEG' TO SDLTR-TEXT
SDL#U      SET   SDLTR-VAR-TYPE-DEC TO TRUE
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         DECIMAL-DATA-SIGNED-NEG
SDL#H      BY CONTENT LENGTH OF DECIMAL-DATA-SIGNED-NEG
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           MOVE  123456789      TO BINARY-DATA
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V9'       TO SDLTR-TYPE
SDL#F      MOVE 'BINARY-DATA' TO SDLTR-TEXT
SDL#U      SET   SDLTR-VAR-TYPE-BIN TO TRUE
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         BINARY-DATA
SDL#H      BY CONTENT LENGTH OF BINARY-DATA
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#Z      MOVE 'STOP '    TO SDLTR-TYPE
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           GOBACK.

       END PROGRAM SAMPLB01.
```

Each inserted statement is marked with the string “SDL#” in the first four positions. Therefore all inserted statements may easily be excluded from view in the ISPF edit with a simple “`x all SDL#`” command. Looking at the included code we see the communication area named SDLTR-PARM which is used to pass the necessary information to SDLTRACE; its structure should not be changed by the application, except of course to set the required parameters, which may change form call to call. The initialization call is executed before any other application program statement and causes the allocation of the trace dataset "*user-id*.SDLAPPL1.COB01JOB.XX.Dyymmdd.ThhmmA". The name is built using the values of the parameters "DSN qualifier" and "Application-ID", followed by the job name under which the program is being run. The "XX" is a two-character qualifier computed using as a base the actual JOB-ID. The DSN is then completed with the current date and time. In addition the environment is set up for all subsequent trace calls. All the information needed for execution in any sub-system, TSO, Batch, CICS, IMS, DB/2, WLM, etc. is provided so that there is no need to change the JCL or any other runtime parameter.

The code contains all statements necessary to produce the trace data when the module is being executed. Since JCL for compilation and linking has been provided as well, the member may be submitted directly from the ISPF editor window. After execution the name of the dynamically allocated trace dataset will be listed in the job output:

```
+SDLTRACE – user-id.SDLAPPL1.COB01JOB.QW.D110314.T0823A
```

and its contents will be the following:

```
START TRACE: SAMPLB01      SDLTRACE VER 4.5   09/13/05          03/14/2015 08:23
V1    CHARACTER-DATA                    Hello, COBOL!
V2    CHARACTER-DATA-LONG           0   This is character data that extends over
V2                                 40 *  more than one line and shows how data i
V2                                 80 * s displayed on several lines
V2                                120 *
       =      3 IDENTICAL LINES - - - - - - - - - - - - - - - - - - - - - - - -
V3    NUMERIC-DATA-UNSIGNED             00000123
V4    NUMERIC-DATA-SIGNED-POS          +00000456
V5    NUMERIC-DATA-SIGNED-NEG          -00000789
V6    DECIMAL-DATA-UNSIGNED           P 0000123
V7    DECIMAL-DATA-SIGNED-POS         P+0000456
V8    DECIMAL-DATA-SIGNED-NEG         P-0000789
V9    BINARY-DATA                     B    123456789
STOP  TRACE: SAMPLB01      SDLTRACE VER 4.5   09/13/05          03/14/2015 08:23
```

The first and the last line show the name of the program being traced, the ***SDLTRACE*** version number and date and time of execution. The other lines trace the path of execution through the program. Each trace line is marked with an identification number consisting of one letter followed by up to four digits. Variables are denoted by "V", and the variable name is displayed following the identification number. The actual data value is then displayed starting in column 40 and formatted according to the variable's type.

Character data is listed without conversion; if it does not fit on one line it will be continued on succeeding lines and marked with an asterisk "*"; in addition the relative offset to the beginning of the continuation line is indicated. If data in adjacent lines is identical it will be displayed only once and the number of identical lines counted. (This check for duplicate lines may be disabled by setting the ATRACE panel switch `Count duplicates` to "`N`", which results in `SET SDLTR-DUPLICAT-OFF TO TRUE` during processing of the COBOL source program).

Numeric data is displayed as shown in the example, unsigned data without a sign and signed data with a "+" or "-" in front. The number of positions indicates the amount specified in the source code definition of the variable. Packed decimal data (COMP-3) is also marked with a "P", with the length always an odd number (though the definition in the source code may have specified an even number). Binary data (COMP) is marked with a "B" and always uses a field of 12 bytes, adjusted to the right, with a "-" sign appended if the number is negative.

Each trace line is actually 133 bytes long, out of which only the first 80 are shown in the example above because all 133 bytes will not fit on one line in this document. The remaining 53 bytes of each line are listed here separately:

```
SAMPLB01     1         1 T 2015-03-14 08:23:36.184826
SAMPLB01     1         2 T 2015-03-14 08:23:36.224077
SAMPLB01     1         3 T 2015-03-14 08:23:36.224088
SAMPLB01     1         4 T 2015-03-14 08:23:36.224088
SAMPLB01     1         5 T 2015-03-14 08:23:36.224088
SAMPLB01     1         6 T 2015-03-14 08:23:36.224088
SAMPLB01     3         9 T 2015-03-14 08:23:36.224088
SAMPLB01     1        10 T 2015-03-14 08:23:36.224095
SAMPLB01     1        11 T 2015-03-14 08:23:36.224100
SAMPLB01     1        12 T 2015-03-14 08:23:36.224104
SAMPLB01     1        13 T 2015-03-14 08:23:36.224108
SAMPLB01     1        14 T 2015-03-14 08:23:36.224113
SAMPLB01     1        15 T 2015-03-14 08:23:36.224117
SAMPLB01     1        16 T 2015-03-14 08:23:36.224121
SAMPLB01     1        17 T 2015-03-14 08:23:36.224125
```

The name of the program that generates the trace line is listed first. It is followed by the duplication counter indicating the number of occurrences of the data line, which in most cases is 1. Lines with identical data in column 40 to 80 are counted and displayed with a message together with the count. In this case the lines 7, 8 and 9 are identical to line 6 and therefore the duplication count is 3. As mentioned before, the check for duplicates can easily be disabled by setting `Count duplicates` to “`N`” in the ATRACE panel.

The next number is the sequence counter of the logical record in the trace dataset, beginning with 1 when the dataset is created initially. This sequence counter will be carried over to succeeding datasets that are automatically allocated when a dataset is full or when for other reasons the current one must be closed. There is a limit for sequence numbers: 2.147.483.647 is the highest possible value, after which counting restarts with 1 again.

The ’T’ indicates that this is a TRACE record (the other possible value being ’L’, denoting a LOG record). The timestamp is formatted by the trace engine which reads the system clock immediately before storing the generated trace record into the trace buffer. Thus for every trace record the time of its creation is stored, which can be used to compute elapsed times between arbitrary events. In one of the following examples this will be explained in more detail.

Please note that the timestamps for the lines with sequence numbers 3 to 9 are identical because they all belong to just one logical record.

Chapter 2. Tracing PERFORMs, labels, etc.

The second example is in member SAMPLB02 and contains two paragraphs which are performed in the main program part:

```
       ID DIVISION.

       PROGRAM-ID  SAMPLB02.

       DATA DIVISION.

       WORKING-STORAGE SECTION.
       01  c-data                   PIC  X(40)  VALUE SPACE.

       PROCEDURE DIVISION.

       MAIN.
           PERFORM Paragraph-1
           DISPLAY c-data
           PERFORM Paragraph-2
           DISPLAY c-data
           GOBACK.
       Paragraph-1.
           MOVE 'Program started' TO c-data
           EXIT.
       Paragraph-2.
           MOVE 'Program ended'   TO c-data
           EXIT.
       END PROGRAM SAMPLB02.
```

When this program is executed it will produce the following output:

```
Program started
Program ended
```

To set up this module for execution with tracing we again go to “*user-id*”.SDLTRACE.DEMO.CNTL and type “ex” in front of ATRACE. The panel being displayed will be the one from the last invocation:

```
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
```

On this panel we change SAMPLB01 to SAMPLB02 and SAMPLBX1 to SAMPLBX2 and then select “**x**” in the top right entry and hit “`Enter`” to have SDLPREP executed in TSO. The result will be the display:

```
SDLTRACE - Version 4.5.23      14 Mar 2015 09:30:58 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=TRACE
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - ------------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      SAMPLB02
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     SAMPLBX2
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      49
SDLTRACE - MOVE .... TO variables:            2
SDLTRACE - Number of lines inserted:        128
SDLTRACE - Number of lines with trace:      177
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

Again the interface to SDLTRACE is inserted into the COBOL source as well as code to trace the two MOVE statements. The resultant module SAMPLBX02 will be displayed in the ISPF editor next, with the “`MAIN.`” part of the procedure division changed to:

```
       .
       .
       MAIN.
           PERFORM Paragraph-1
           DISPLAY c-data
           PERFORM Paragraph-2
           DISPLAY c-data
SDL#***------------------------------------------------------------***
SDL#Z      MOVE 'STOP '    TO SDLTR-TYPE
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           GOBACK.
       Paragraph-1.
           MOVE 'Program started' TO c-data
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'c-data' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         c-data
SDL#H      BY CONTENT LENGTH OF c-data
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           EXIT.
       Paragraph-2.
           MOVE 'Program ended'   TO c-data
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V2'       TO SDLTR-TYPE
SDL#F      MOVE 'c-data' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         c-data
SDL#H      BY CONTENT LENGTH OF c-data
```

```
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           EXIT.
       END PROGRAM SAMPLB02.
```

When the above program is run a trace dataset with the following contents will be generated:

```
START TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:34
V1    c-data                            Program started
V2    c-data                            Program ended
STOP  TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:34
```

The two MOVE statements are executed and the trace shows `V1` and `V2` with the variable name `c-data` and the respective values. In order to see the complete flow through the program as it is run we now enlarge the scope of the trace by changing the relevant entries on the ATRACE panel:

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

The three entries `Trace PERFORM`, `Trace PERFORM end` and `Trace labels` are changed to `Y`, and the messages after execution of the panel (just enter “x” in the panel command field) will be:

```
SDLTRACE - Version 4.5.23      14 Mar 2015 09:36:56 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=TRACE
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> PERFORM statements
SDLTRACE - -> Return from PERFORM block
SDLTRACE - -> Labels being encountered
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - ------------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      SAMPLB02
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     SAMPLBX2
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      49
SDLTRACE - PERFORMs to trace:                 2
SDLTRACE - Labels to trace:                   3
SDLTRACE - MOVE .... TO variables:            2
SDLTRACE - Number of lines inserted:        163
SDLTRACE - Number of lines with trace:      212
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

There are two `PERFORM` statements to trace, three `Labels (MAIN, Paragraph1` and `Paragraph2)` and of course the two `MOVE` statements. The changed program now reads as follows, starting at paragraph `MAIN`:

```
       MAIN.
SDL#***------------------------------------------------------------***
```

```
SDL#L      MOVE 'L1'       TO SDLTR-TYPE
SDL#T      MOVE 'MAIN.' TO SDLTR-TEXT
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL.
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#B      MOVE 'P1'       TO SDLTR-TYPE
SDL#T      MOVE 'Paragraph-1' TO SDLTR-TEXT
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           PERFORM Paragraph-1
SDL#***------------------------------------------------------------***
SDL#X      MOVE 'X1'       TO SDLTR-TYPE
SDL#T      MOVE 'Paragraph-1' TO SDLTR-TEXT
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           DISPLAY c-data
SDL#***------------------------------------------------------------***
SDL#B      MOVE 'P2'       TO SDLTR-TYPE
SDL#T      MOVE 'Paragraph-2' TO SDLTR-TEXT
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           PERFORM Paragraph-2
SDL#***------------------------------------------------------------***
SDL#X      MOVE 'X2'       TO SDLTR-TYPE
SDL#T      MOVE 'Paragraph-2' TO SDLTR-TEXT
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           DISPLAY c-data
SDL#***------------------------------------------------------------***
SDL#Z      MOVE 'STOP '    TO SDLTR-TYPE
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           GOBACK.
       Paragraph-1.
SDL#***------------------------------------------------------------***
SDL#L      MOVE 'L2'       TO SDLTR-TYPE
SDL#T      MOVE 'Paragraph-1.' TO SDLTR-TEXT
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL.
SDL#***------------------------------------------------------------***
           MOVE 'Program started' TO c-data
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'c-data' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         c-data
SDL#H      BY CONTENT LENGTH OF c-data
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
           EXIT.
       Paragraph-2.
SDL#***------------------------------------------------------------***
SDL#L      MOVE 'L3'       TO SDLTR-TYPE
SDL#T      MOVE 'Paragraph-2.' TO SDLTR-TEXT
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL.
SDL#***------------------------------------------------------------***
           MOVE 'Program ended'   TO c-data
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V2'       TO SDLTR-TYPE
SDL#F      MOVE 'c-data' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         c-data
SDL#H      BY CONTENT LENGTH OF c-data
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
```

```
            EXIT.
        END PROGRAM SAMPLB02.
```

Execution of this program will result in a dataset with the following contents:

```
START TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:42
L1    ------- MAIN.
P1    PERFORM Paragraph-1
L2    ------- Paragraph-1.
V1    c-data                            Program started
X1    --END-- Paragraph-1
P2    PERFORM Paragraph-2
L3    ------- Paragraph-2.
V2    c-data                            Program ended
X2    --END-- Paragraph-2
STOP  TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:42
```

The first and the last line are generated by SDLTRACE and report the program name SAMPLB02, which was executed on 03/14/2015 at 09:42. Line 2 starts with the indicator “L”, indicating a label; since it is the first one its sequence number is “1”, hence “L1”. This is followed by the label string “-------“ and the actual name “MAIN.”. On the next line we have identifier “P1” denoting the first PERFORM statement, naming Paragraph-1 as target. This is then shown on the following line as “L2“, the second label in the program. There is just one move statement in Paragraph-1 assigning the data “Program started “ to the variable “c-data” identified by “V1“. The program now has reached the end of Paragraph-1 and returns to “MAIN“, which is marked with “X1“ to signal the exit, with the string “--END--“ followed by the name of the perform that is ending, namely “Paragraph-1“.

Similarly Paragraph-2 is traced with the second move statement identified by “V2“, even though the name of the variable “c-data“ is the same as in Paragraph-1. Each occurrence of a traced item is identified by a unique 1 to 4-digit number, preceded by a letter denoting the type (L, P, V, X, etc.). It is therefore easy to find the place in a program where a certain variable is assigned even if it occurs many times in the program. With a simple “f V…” in the ISPF editor the location can be found immediately.

With SDLTRACE it is easy to verify that a program does exactly what is intended. As an example we now change the line:

```
        Paragraph-1.
```

in program SAMPLB02 in library “*user-id”*.SDLTRACE.DEMO.COBOL to read:

```
        Paragraph-1 SECTION.
```

In order to do that please go to library “*user-id”*.SDLTRACE.DEMO.COBOL and edit SAMPLB02 with the SPF editor. The Procedure Division of the program SAMPLB02 now is:

```
        MAIN.
            PERFORM Paragraph-1
            DISPLAY c-data
            PERFORM Paragraph-2
            DISPLAY c-data
            GOBACK.
        Paragraph-1 SECTION.
            MOVE 'Program started' TO c-data
            EXIT.
        Paragraph-2.
            MOVE 'Program ended'   TO c-data
            EXIT.
        END PROGRAM SAMPLB02.
```

After the change the member should be saved by hitting PF3.

Execution of this program will produce the following output:

```
Program ended
Program ended
```

Why that? To a knowledgeable COBOL programmer this is exactly what should be expected; to others the result is somewhat puzzling. A simple trace, however, reveals what has happened. After executing the panel ATRACE again and then compiling and running the program we will get the trace output:

```
START TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:49
L1    ------- MAIN.
P1    PERFORM Paragraph-1
L2    ------- Paragraph-1 SECTION.
V1    c-data                          Program started
L3    ------- Paragraph-2.
V2    c-data                          Program ended
X1    --END-- Paragraph-1
P2    PERFORM Paragraph-2
L3    ------- Paragraph-2.
V2    c-data                          Program ended
X2    --END-- Paragraph-2
STOP  TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:49
```

According to the COBOL language specification this is exactly what the program should do: A section ends when a new section starts or at the end of the program; the `EXIT` statement has no effect. Thus the program “falls through” and executes Paragraph-2 twice, once as part of “SECTION” Paragraph-1 and then as single paragraph within “SECTION” Paragraph-1. It is not what the programmer, presumably, intended, but it nevertheless is correct and shows how important verification of program logic really is.

Chapter 3. Tracing variables with indices

The next program we look at is SAMPLB03 in library “user-id“.SDLTRACE.DEMO.COBOL:

```
ID DIVISION.

PROGRAM-ID  SAMPLB03.

DATA DIVISION.

WORKING-STORAGE SECTION.
01  c-const        PIC X(26) VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
01  c-data.
  10 c-data-row              OCCURS 3.
    20 c-data-item PIC X(05) OCCURS 3.
01  I              PIC 9(05) BINARY.
01  J              PIC 9(05) BINARY.
PROCEDURE DIVISION.
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3
      PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
        MOVE c-const(I:J)      TO c-data-item(I J)
      END-PERFORM
    END-PERFORM
    GOBACK.
END PROGRAM SAMPLB03.
```

The two-dimensional array `c-data-item` is initialized with substrings from the string `c-const` containing the alphabet. The indices `I` and `J` are varied from 1 to 3 and are used to select the substrings as well. To verify the operation of this program we again call up ATRACE out of library *“user-id”*.SDLTRACE.DEMO.CNTL and specify SAMPLB03 as input and SAMPLBX3 as output member. In addition the three parameters “Trace PERFORM”, “Trace PERFORM end” and “Trace labels” should be reset to “N”. The panel should thus look as follows:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                             Screen 3 of 3
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         SAMPLB03
Output member        SAMPLBX3        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
```

Execution of this panel will create SAMPLBX3, which, when run, will produce the following trace dataset:

```
START TRACE: SAMPLB03      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:55
V1    c-data-item(I J)                  A                   1            1
V1    c-data-item(I J)                  AB                  1            2
V1    c-data-item(I J)                  ABC                 1            3
V1    c-data-item(I J)                  B                   2            1
V1    c-data-item(I J)                  BC                  2            2
V1    c-data-item(I J)                  BCD                 2            3
V1    c-data-item(I J)                  C                   3            1
V1    c-data-item(I J)                  CD                  3            2
V1    c-data-item(I J)                  CDE                 3            3
STOP  TRACE: SAMPLB03      SDLTRACE VER 4.5   09/13/05          03/14/2015 09:55
```

The trace shows that the program produces the expected result: The substrings with length 1, 2 and 3 are assigned to the proper array elements, indicated by the respective indices whose values are listed together with the value of the array variable (up to two indices are traced automatically). If there are more than two, then only the first two are traced and the remaining indices are simply ignored (for tracing purposes).

Chapter 4. Measuring execution times

Each trace record contains in its rightmost part the actual timestamp with microsecond resolution so that the elapsed time between any two events can easily be computed. However, instead of doing that manually the trace program can be used to compute these differences. In addition to the elapsed time, SDLTRACE also determines the amount of CPU time used during each interval. Both values are recorded in the trace dataset following each trace record. The module SAMPLB04 shows how this feature can be used to easily get execution times:

```
ID DIVISION.

PROGRAM-ID  SAMPLB04.

DATA DIVISION.

WORKING-STORAGE SECTION.

01  WAIT                   PIC  X(8)  VALUE  'SDLWAIT'.
01  WAIT-PARM.
  05                       PIC S9(4)  COMP  VALUE 8.
  05  WAIT-TIME.
    10  WAIT-TIME-HH       PIC  9(2)  VALUE  ZERO.
    10  WAIT-TIME-MM       PIC  9(2)  VALUE  ZERO.
    10  WAIT-TIME-SS       PIC  9(2)  VALUE  ZERO.
    10  WAIT-TIME-TH       PIC  9(2)  VALUE  ZERO.
01  I                      PIC S9(3)  COMP-3.

PROCEDURE DIVISION.
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3
      MOVE  I      TO     WAIT-TIME-SS
      CALL  WAIT   USING  WAIT-PARM  END-CALL
    END-PERFORM
    MOVE ZERO TO RETURN-CODE
    GOBACK.

END PROGRAM SAMPLB04.
```

In SAMPLB04 the service module SDLWAIT is called 3 times, with the parameter set to 1, 2 and then 3 seconds. (The purpose of SDLWAIT is to introduce delays into the execution of programs, with the amount of time to wait specified in the parameter block). Again the trace code is inserted by calling up the panel ATRACE:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                                 Screen 4 of 4
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         SAMPLB04
Output member        SAMPLBX4        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
```

After executing this panel the Procedure Division of SAMPLB04 (now in member SAMPLBX4) will be:

```
       PROCEDURE DIVISION.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'SAMPLB04' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE 'SDLAPPL1' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 5          TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-OFF  TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-ON   TO TRUE
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL.
SDL#***------------------------------------------------------------***

           PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3
             MOVE  I      TO     WAIT-TIME-SS
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'WAIT-TIME-SS' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         WAIT-TIME-SS
SDL#H      BY CONTENT LENGTH OF WAIT-TIME-SS
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V2'       TO SDLTR-TYPE
SDL#F      MOVE '------> CALL WAIT' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
             CALL  WAIT   USING  WAIT-PARM  END-CALL
           END-PERFORM
           MOVE ZERO TO RETURN-CODE
SDL#***------------------------------------------------------------***
SDL#I      MOVE 'V3'       TO SDLTR-TYPE
SDL#F      MOVE 'RETURN-CODE' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         RETURN-CODE
SDL#H      BY CONTENT LENGTH OF RETURN-CODE
SDL#E      END-CALL
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#Z      MOVE 'STOP '    TO SDLTR-TYPE
```

```
SDL#C      CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#***------------------------------------------------------------***
           GOBACK.

       END PROGRAM SAMPLB04.
```

Running this program will produce the following trace dataset:

```
START TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:02
V1    WAIT-TIME-SS                      01
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      02
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      03
V2    ------> CALL WAIT
V3    RETURN-CODE                     X'0000'
STOP  TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:02
```

with the rightmost 53 columns:

```
                           SAMPLB04     1         1 T 2015-03-14 10:02:21.101073
                           SAMPLB04     1         2 T 2015-03-14 10:02:21.138513
                           SAMPLB04     1         3 T 2015-03-14 10:02:21.138539
                           SAMPLB04     1         4 T 2015-03-14 10:02:22.140225
                           SAMPLB04     1         5 T 2015-03-14 10:02:22.140232
                           SAMPLB04     1         6 T 2015-03-14 10:02:24.140312
                           SAMPLB04     1         7 T 2015-03-14 10:02:24.140322
                           SAMPLB04     1         8 T 2015-03-14 10:02:27.140368
                           SAMPLB04     1         9 T 2015-03-14 10:02:27.140378
```

The times for the first MOVE and the CALL statements are in line 3, 4, 6 and 8 as follows: `10:02:21`, `10:02:22`, `10:02:24` and `10:02:27`. This shows that the delays are executed properly and amount to 1, 2 and 3 seconds as requested.

In the lower left part of the panel ATRACE there are the lines:

```
Enable timing        N (Y/N)
Timing threshold ____0 (0-32767 ms)
```

These two entries control the timing option of SDLTRACE. By setting `Enable timing` to `Y` and `Timing threshold` to `0` the elapsed time and the CPU time for each trace record will be computed. Please set the parameter `Enable timing` to `Y` and execute the panel again. There will be a slight change in the generated module SAMPLBX4 in the trace initialization paragraph where instead of

```
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
```

we now have:

```
SDL#K      SET  SDLTR-TIMING-ON     TO TRUE
```

When run with this change the program will generate the following trace dataset:

```
START TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:14
V1    WAIT-TIME-SS                      01
V2                     Measured elapsed clock time:               10 microseconds
V2                     z/OS recorded TASK CPU time:                0 microseconds
V2    ------> CALL WAIT
V1                                      Clock time:        1.001708 seconds
V1                                        CPU time:              130 microseconds
V1    WAIT-TIME-SS                      02
V2                                      Clock time:               11 microseconds
```

```
V2                                        CPU time:               0 microseconds
V2    ------> CALL WAIT
V1                                      Clock time:        2.000077 seconds
V1                                        CPU time:              33 microseconds
V1    WAIT-TIME-SS                      03
V2                                      Clock time:              12 microseconds
V2                                        CPU time:               0 microseconds
V2    ------> CALL WAIT
V3                                      Clock time:        3.000041 seconds
V3                                        CPU time:              36 microseconds
V3    RETURN-CODE                     X'0000'
STOP  TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:14
```

There are two additional lines indicating different times: ***Measured elapsed clock time,*** which is the difference between the relevant timestamps, and: ***z/OS recorded TASK CPU time,*** which is the corresponding CPU time as attributed to the current task by the operating system. Since the CPU time is computed only when a task change takes place there are many instances where the displayed CPU time is zero because the current task is still active and the corresponding entry in the TCB has not yet been updated by the system. Therefore the individual CPU time values are not very useful per se, except as an indicator that the task has not been interrupted, and as a result the elapsed time also represents CPU usage if the corresponding CPU time value indicates 0.

The amount of timing information may be limited by specifying a certain threshold so that only elapsed times above that value are recorded. Please call up ATRACE again for member SAMPLB04 and change the following entries to read:

```
Enable timing        Y (Y/N)
Timing threshold _3000 (0-32767 ms)
```

This will limit the timing output to values above 3 seconds (3000 ms). The trace file of SAMPLBX4 will show the following:

```
START TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:17
V1    WAIT-TIME-SS                      01
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      02
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      03
V2    ------> CALL WAIT
V3                     Measured elapsed clock time:        3.000052 seconds
V3                     z/OS recorded TASK CPU time:              33 microseconds
V3    RETURN-CODE                     X'0000'
STOP  TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:17
```

As requested, only elapsed times exceeding 3 seconds are recorded. In this case it is the call to the subroutine WAIT with parameter 03 that satisfies the condition.

The threshold is specified in milliseconds. The minimum positive value that may be specified is 1ms. Since most statements in a program execute in less than that only calls to subroutines or accesses to external databases will be recorded if 1ms is specified.

For a detailed time consumption analysis of a program or of a complete application consisting of many individual modules, the **logging mode** of SDLTRACE provides all necessary tools to determine where time is spent and how much is used by each component. It is therefore recommended to use the logging mode instead of tracing when analyzing the time consumption distribution of an application.

Chapter 5. Controlling the allocation of trace datasets

There are five parameters to control the allocation of trace datasets: DSN qualifier, Application-ID, JOB-ID check, DSN alloc space and DSN alloc time. These parameters are specified on the ATRACE panel in the section:

```
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)   0___ (0-1440)   Trace variables    Y (Y/N)
```

1. DSN qualifier

The `DSN qualifier` specifies the high level of the trace dataset that is automatically allocated when a trace starts. It may be any name which is a valid DSN and for which the job has the authorization (in RACF or similar systems) for allocation. Within all sample jobs we use the current user's ID which of course does have this authorization when the jobs are submitted by TSO. If the program containing trace code is to be executed in another environment, for example in CICS, IMS, WLM, etc., then the high level should be one that is authorized for "Write access" in those systems.

The `DSN qualifier` is not limited to just one level; it could also be `XYZ.TEST`, for example (if allocation authority for XYZ exists). In case this authorization is missing it will not be possible to allocate the trace dataset and an error message will be issued. No trace data will be produced in this case; the program, however, will be executed as if no trace code were present.

2. Application-ID

The `Application-ID` determines the second level of the trace dataset name. It may be any valid DSN and is used to distinguish between different programs or a set of programs belonging to a group. If in one job several programs with trace code are executed and the application-ID is the same then the trace data is recorded in a single dataset. Thus the call of a subroutine causes the trace data of that subroutine to be listed immediately after the call in the same dataset. For an example please look at program SAMPLB05 in library "user-id".SDLTRACE.DEMO.COBOL:

```
ID DIVISION.

PROGRAM-ID  SAMPLB05.

DATA DIVISION.

WORKING-STORAGE SECTION.
01  pgm                       pic  x(8)  value space.

PROCEDURE DIVISION.
    move 'SAMPLB01' to pgm
    call pgm
    move 'SAMPLB02' to pgm
    call pgm
    move 'SAMPLB03' to pgm
    call pgm
    move 'SAMPLB04' to pgm
    call pgm
    goback.
END PROGRAM SAMPLB05.
```

The program calls the four samples we have used so far. The load modules which are called when this program is run already have trace code inserted from executing the samples as described above. Execution of SAMPLB05 will therefore result in a trace file that contains the trace data of all four programs combined in one dataset. Just submit SAMPLB05 and look at the generated file. It will show the following:

```
START TRACE: SAMPLB01      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
V1    CHARACTER-DATA                    Hello, COBOL!
V2    CHARACTER-DATA-LONG           0   This is character data that extends over
V2                                 40 *  more than one line and shows how data i
V2                                 80 * s displayed on several lines
V2                                120 *
      =       3 IDENTICAL LINES - - - - - - - - - - - - - - - - - - - - - - - -
V3    NUMERIC-DATA-UNSIGNED             00000123
V4    NUMERIC-DATA-SIGNED-POS          +00000456
V5    NUMERIC-DATA-SIGNED-NEG          -00000789
V6    DECIMAL-DATA-UNSIGNED           P 0000123
V7    DECIMAL-DATA-SIGNED-POS         P+0000456
V8    DECIMAL-DATA-SIGNED-NEG         P-0000789
V9    BINARY-DATA                     B    123456789
STOP  TRACE: SAMPLB01      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
START TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
L1    ------- MAIN.
P1    PERFORM Paragraph-1
L2    ------- Paragraph-1 SECTION.
V1    c-data                            Program started
L3    ------- Paragraph-2.
V2    c-data                            Program ended
X1    --END-- Paragraph-1
P2    PERFORM Paragraph-2
L3    ------- Paragraph-2.
V2    c-data                            Program ended
X2    --END-- Paragraph-2
STOP  TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
START TRACE: SAMPLB03      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
V1    c-data-item(I J)                  A                   1            1
V1    c-data-item(I J)                  AB                  1            2
V1    c-data-item(I J)                  ABC                 1            3
V1    c-data-item(I J)                  B                   2            1
V1    c-data-item(I J)                  BC                  2            2
V1    c-data-item(I J)                  BCD                 2            3
V1    c-data-item(I J)                  C                   3            1
V1    c-data-item(I J)                  CD                  3            2
V1    c-data-item(I J)                  CDE                 3            3
STOP  TRACE: SAMPLB03      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
START TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
V1    WAIT-TIME-SS                      01
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      02
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      03
V2    ------> CALL WAIT
V3                       Measured elapsed clock time:       3.000058 seconds
V3                       z/OS recorded TASK CPU time:             29 microseconds
V3    RETURN-CODE                     X'0000'
STOP  TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:25
```

In order to trace SAMPLB05, too, please call up ATRACE again and set up the program for tracing by entering the following values in the panel:

```
                                                                   Screen 5 of 5
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         SAMPLB05
Output member        SAMPLBX5        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
```

Executing the panel above will insert trace code into SAMPLB05, too, so that all five sample programs now contain calls to SDLTRACE. When run SAMPLB05 will generate the following file:

```
START TRACE: SAMPLB05      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
V1    pgm                                SAMPLB01
V2    ------> CALL pgm
START TRACE: SAMPLB01      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
V1    CHARACTER-DATA                     Hello, COBOL!
V2    CHARACTER-DATA-LONG            0   This is character data that extends over
V2                                  40 *  more than one line and shows how data i
V2                                  80 * s displayed on several lines
V2                                 120 *
      =       3 IDENTICAL LINES - - - - - - - - - - - - - - - - - - - - - - - - -
V3    NUMERIC-DATA-UNSIGNED              00000123
V4    NUMERIC-DATA-SIGNED-POS           +00000456
V5    NUMERIC-DATA-SIGNED-NEG           -00000789
V6    DECIMAL-DATA-UNSIGNED            P 0000123
V7    DECIMAL-DATA-SIGNED-POS          P+0000456
V8    DECIMAL-DATA-SIGNED-NEG          P-0000789
V9    BINARY-DATA                      B    123456789
STOP  TRACE: SAMPLB01      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
V3    pgm                                SAMPLB02
V4    ------> CALL pgm
START TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
L1    ------- MAIN.
P1    PERFORM Paragraph-1
L2    ------- Paragraph-1 SECTION.
V1    c-data                             Program started
L3    ------- Paragraph-2.
V2    c-data                             Program ended
X1    --END-- Paragraph-1
P2    PERFORM Paragraph-2
L3    ------- Paragraph-2.
V2    c-data                             Program ended
X2    --END-- Paragraph-2
STOP  TRACE: SAMPLB02      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
V5    pgm                                SAMPLB03
V6    ------> CALL pgm
START TRACE: SAMPLB03      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
```

```
V1    c-data-item(I J)                  A                   1            1
V1    c-data-item(I J)                  AB                  1            2
V1    c-data-item(I J)                  ABC                 1            3
V1    c-data-item(I J)                  B                   2            1
V1    c-data-item(I J)                  BC                  2            2
V1    c-data-item(I J)                  BCD                 2            3
V1    c-data-item(I J)                  C                   3            1
V1    c-data-item(I J)                  CD                  3            2
V1    c-data-item(I J)                  CDE                 3            3
STOP  TRACE: SAMPLB03      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
V7    pgm                               SAMPLB04
V8    ------> CALL pgm
START TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
V1    WAIT-TIME-SS                      01
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      02
V2    ------> CALL WAIT
V1    WAIT-TIME-SS                      03
V2    ------> CALL WAIT
V3                     Measured elapsed clock time:        3.000045 seconds
V3                     z/OS recorded TASK CPU time:              29 microseconds
V3    RETURN-CODE                     X'0000'
STOP  TRACE: SAMPLB04      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
STOP  TRACE: SAMPLB05      SDLTRACE VER 4.5   09/13/05          03/14/2015 10:34
```

The rightmost part of the 133-byte trace lines for the data above is:

```
                           SAMPLB05     1          1 T 2015-03-14 10:34:49.567871
                           SAMPLB05     1          2 T 2015-03-14 10:34:49.606044
                           SAMPLB05     1          3 T 2015-03-14 10:34:49.606054
                           SAMPLB01     1          4 T 2015-03-14 10:34:49.607969
                           SAMPLB01     1          5 T 2015-03-14 10:34:49.607980
                           SAMPLB01     1          6 T 2015-03-14 10:34:49.607984
                           SAMPLB01     1          7 T 2015-03-14 10:34:49.607984
                           SAMPLB01     1          8 T 2015-03-14 10:34:49.607984
                           SAMPLB01     1          9 T 2015-03-14 10:34:49.607984
                           SAMPLB01     3         12 T 2015-03-14 10:34:49.607984
                           SAMPLB01     1         13 T 2015-03-14 10:34:49.607992
                           SAMPLB01     1         14 T 2015-03-14 10:34:49.607997
                           SAMPLB01     1         15 T 2015-03-14 10:34:49.608001
                           SAMPLB01     1         16 T 2015-03-14 10:34:49.608005
                           SAMPLB01     1         17 T 2015-03-14 10:34:49.608009
                           SAMPLB01     1         18 T 2015-03-14 10:34:49.608013
                           SAMPLB01     1         19 T 2015-03-14 10:34:49.608018
                           SAMPLB01     1         20 T 2015-03-14 10:34:49.608023
                           SAMPLB05     1         21 T 2015-03-14 10:34:49.610570
                           SAMPLB05     1         22 T 2015-03-14 10:34:49.618091
                           SAMPLB02     1         23 T 2015-03-14 10:34:49.620024
                           SAMPLB02     1         24 T 2015-03-14 10:34:49.620035
                           SAMPLB02     1         25 T 2015-03-14 10:34:49.620039
                           SAMPLB02     1         26 T 2015-03-14 10:34:49.620043
                           SAMPLB02     1         27 T 2015-03-14 10:34:49.620047
                           SAMPLB02     1         28 T 2015-03-14 10:34:49.620052
                           SAMPLB02     1         29 T 2015-03-14 10:34:49.620056
                           SAMPLB02     1         30 T 2015-03-14 10:34:49.620060
                           SAMPLB02     1         31 T 2015-03-14 10:34:49.621313
                           SAMPLB02     1         32 T 2015-03-14 10:34:49.621318
                           SAMPLB02     1         33 T 2015-03-14 10:34:49.621322
                           SAMPLB02     1         34 T 2015-03-14 10:34:49.621326
                           SAMPLB02     1         35 T 2015-03-14 10:34:49.621331
                           SAMPLB05     1         36 T 2015-03-14 10:34:49.623761
                           SAMPLB05     1         37 T 2015-03-14 10:34:49.630553
                           SAMPLB03     1         38 T 2015-03-14 10:34:49.632426
                           SAMPLB03     1         39 T 2015-03-14 10:34:49.632437
```

```
SAMPLB03     1         40 T 2015-03-14 10:34:49.632442
SAMPLB03     1         41 T 2015-03-14 10:34:49.632447
SAMPLB03     1         42 T 2015-03-14 10:34:49.632451
SAMPLB03     1         43 T 2015-03-14 10:34:49.632455
SAMPLB03     1         44 T 2015-03-14 10:34:49.632460
SAMPLB03     1         45 T 2015-03-14 10:34:49.632464
SAMPLB03     1         46 T 2015-03-14 10:34:49.632469
SAMPLB03     1         47 T 2015-03-14 10:34:49.632473
SAMPLB03     1         48 T 2015-03-14 10:34:49.632478
SAMPLB05     1         49 T 2015-03-14 10:34:49.634930
SAMPLB05     1         50 T 2015-03-14 10:34:49.643433
SAMPLB04     1         51 T 2015-03-14 10:34:49.645524
SAMPLB04     1         52 T 2015-03-14 10:34:49.645535
SAMPLB04     1         53 T 2015-03-14 10:34:49.645540
SAMPLB04     1         54 T 2015-03-14 10:34:50.648872
SAMPLB04     1         55 T 2015-03-14 10:34:50.648881
SAMPLB04     1         56 T 2015-03-14 10:34:52.648974
SAMPLB04     1         57 T 2015-03-14 10:34:52.648982
SAMPLB04     1         58.T 2015-03-14 10:34:55.649027
SAMPLB04     1         58.T 2015-03-14 10:34:55.649027
SAMPLB04     1         58 T 2015-03-14 10:34:55.649027
SAMPLB04     1         59 T 2015-03-14 10:34:55.649039
SAMPLB05     1         60 T 2015-03-14 10:34:55.651847
```

Trace lines from SAMPLB05 are highlighted to distinguish them from the output of the called programs. They show the individual calls and immediately following a trace of the execution of the called module. Every program carries its own trace parameters and options, that is, each trace definition for a module with ATRACE is local to that module. This can be seen above where in SAMPLB02 tracing of paragraphs is enabled, and only SAMPLB04 includes timing information. A called program containing trace code may in turn call another program with trace code; there is no limit to the number of nested calls.

If an application-ID is not specified in the ATRACE panel then the name of the program is taken and the trace dataset name is generated accordingly. By specifying application-IDs depending on functional differences individual trace datasets can easily be generated for groups of programs or individual modules.

3. JOB-ID check

The parameter `JOB-ID check` is used to limit tracing to certain jobs or monitors. An asterisk ‘*’ means that a trace should be generated irrespective of the job or monitor name under which it is executed. If a name is specified then a trace is produced only if the actual job or monitor is equal to that name. If the first part of a name with an asterisk appended (as in ‘ABC*’) is specified then a trace is generated only for jobs or monitors whose names start with that string. For all other jobs no trace is produced and the application runs as if no trace code existed, except that the return-code is set to 4. (To preserve an application’s return-code just set the option `Save RETURN-CODE` to ‘`Y`’. This should generally be set if the return code from SDLTRACE is not checked by the application. Since the invocation of SDLTRACE is an external call 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 is therefore the recommended setting).

Instead of a single jobname it is also possible to provide a list of names for which tracing is to be enabled. This list is specified in the Assembler module SDLJOBLG in library *“user-id”.SDLTRACE.DEMO.ASM.* For information regarding its use, please see the comments in that module. To enable the use of the list the program SDLJOBLG must be linked into SDLTRACE and the option `JOB-ID check` must be set to the string “`JOB-LIST`“ in the ATRACE panel.

4. DSN alloc (tracks)

The space parameter `DSN alloc (tracks)` specifies the number of tracks between 1 and 9999 to be allocated when a trace dataset is required. The amount of tracks depends on the type of application. Independent of the allocation size 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’ 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’.

As an example the program SAMPLB06, which is simply the first sample program executed several times in a loop, can be set up for tracing with the following parameters:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                              Screen 6 of 6
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         SAMPLB06
Output member        SAMPLBX6       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)   1___ (1-9999)  Trace labels       N (Y/N)
DSN time (minutes)   0___ (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     Y (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
```

The number of tracks to be allocated (parameter `DSN alloc (tracks)`) has been set to 1, so as to force the allocation of a second dataset after relatively few trace records have been written. With the loop counter set to 100, a third dataset will be allocated too, so that in the job output we will have:

```
+SDLTRACE - user-id.SDLAPPL1.COB06JOB.FP.D110315.T0930A
+SDLTRACE - user-id.SDLAPPL1.COB06JOB.FP.D110315.T0930B
+SDLTRACE - user-id.SDLAPPL1.COB06JOB.FP.D110315.T0930C
```

As already mentioned, trace data will be written alternatively on datasets ‘B’ and ‘C’ until the program terminates or until the time parameter, described next, forces the allocation of a new dataset.

5. DSN time (minutes)

The parameter `DSN time (minutes)` specifies the time interval between 1 and 1440 minutes during which a trace dataset should be active. 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 tracing continues with recording on a new dataset with the last letter ‘A’, and then ‘B’ and ‘C’. 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 `DSN time` 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 at 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 trace 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 at 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.

There are many other options available for tracing which are described in detail in the reference manual. The information provided so far, however, should be enough to begin using SDLTRACE with some of your own programs. You might be surprised to see what your code really does, and if it is exactly what you intended it to do (and the Compiler faithfully translated that into executable instructions), then you can be sure that the programs are not just accidentally delivering the expected results.

Introduction to Logging Facility for COBOL

The logging facility of SDLTRACE is a special subset of the trace system and is used to record events in real-time without any measurable impact on performance of the application. Just as the trace facility, the logging mode runs in any execution environment (Batch, CICS, IMS, DB2, WLM, TSO, etc.) using a simple CALL interface. On current (2011) mainframe machines it takes about 15 microseconds to record a single event, so that for a million log records only 15 CPU seconds are needed. The collection of the data on the various images in a multi-SYSPLEX environment and the consolidation and management of the log data is fully automated with the use of SDLMERGE, a separate program that can be set up to run as a permanent job (similar to a Started Task, but easier to operate).

Only standard interfaces are being used so that any application programmer can set up logging for his or her application. The only requirement is that the job or monitor under which logging is performed has the authority (RACF or similar systems) to write to the dynamically allocated datasets, where the high levels of the datasets are specified by the user at setup time.

Chapter 1. Preparing a COBOL program for logging

There are several small programs provided in the COBOL library to show how to use the logging facility. The examples all have JCL statements included so that the modules can be executed right away.
The job cards of the members in dataset “*user-id”.*SDLTRACE.DEMO.COBOL may have to be edited before submitting the jobs, for example to insert accounting information required by your installation.

In the example module LOGMOD11 in library *user-id.*SDLTRACE.DEMO.COBOL we will insert all code necessary to produce a log whenever the module is called. The COBOL code of the skeleton program is as follows:

```
Identification Division.

Program-ID. LOGMOD11.

Data Division.
Working-Storage Section.
01 Log-Record          pic x(80).

Linkage Section.
01 Log-Data            pic x(80).

Procedure Division using Log-Data.
Main.
    move Log-Data to Log-Record
    goback.

End program LOGMOD11.
```

This program just moves the 80-byte field supplied by the calling program to a local variable and then returns. This in itself does not seem to be very useful. However, the code serves as a base for the “real” logging module to be automatically created in the next step.

To extend this skeleton module with code for execution with logging just go into ISPF panel 3.4, display the members of *user-id.*SDLTRACE.DEMO.CNTL, step down to member ATRACE and type “**ex**” (short for “exec”) in front of it. A panel similar to the following one will be displayed:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                      Screen 6 of 6
```

```
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         LOGMOD11
Output member        LOGMXD11        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (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)   60__ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     N (Y/N)         Include string #1  LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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
```

The values displayed in the input fields will probably be different in some locations from the ones listed above. Please make sure that they are set to exactly the values shown here, especially the field `Trace/Log mode` which should be set to “`L`”, and the field `Include string #1` which should show “`LOG-RECORD` ”, the name of the variable into which the data to be logged will be moved.

To process this panel, please enter “x” in the top right field and hit the “`Enter`” key. The following messages will be generated for the input values on the panel above, (provided that the values in your panel which you submit for execution are identical to those shown above):

```
SDLTRACE - Version 4.5.23      15 Mar 2015 10:02:04 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - -----------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD11
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD11
SDLTRACE - -----------------------------------
SDLTRACE - Number of lines in original:      39
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      166
SDLTRACE - -----------------------------------
SDLTRACE - End of process  RC=0
```

Hitting the “`Enter`” key again will display the changed module LOGMXD11 in SPF edit mode:

```
       Identification Division.

       Program-ID. LOGMOD11.

       Data Division.
       Working-storage Section.
SDL#***-----------------------------------------------------------***
SDL#Y  01  SDLTR-PARM  GLOBAL.
SDL#Y      05  SDLTR-VALIDCHK-A   PIC X(8)   VALUE  'SDLTRACE'.
```

```
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VERSION      PIC X(8)   VALUE  'VER 4.5 '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-DSN-HILEVEL  PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-APPL-ID      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PGMNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-JOBNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TYPE         PIC X(5)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TEXT         PIC X(50)  VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-START   PIC X(8)   VALUE  'PERFORM '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-END     PIC X(8)   VALUE  '--END-- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-LABEL        PIC X(8)   VALUE  '------- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PRI-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-SEC-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-VAR-LENGTH   PIC 9(4)   VALUE 31   BINARY.
SDL#Y      05  SDLTR-FLD-LENGTH   PIC 9(4)   VALUE 13   BINARY.
SDL#Y      05  SDLTR-THRESHOLD    PIC 9(4)   VALUE 0    BINARY.
SDL#Y      05  SDLTR-NEWTIM       PIC 9(4)   VALUE 1440 BINARY.
SDL#Y      05  SDLTR-RETN-CODE    PIC 9(2)   VALUE   0.
SDL#Y      05  SDLTR-RETN-CBIN    PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TRACE        PIC 9(1)   VALUE   1.
SDL#Y        88 SDLTR-TRACE-ON               VALUE   1.
SDL#Y        88 SDLTR-TRACE-OFF              VALUE   0.
SDL#Y      05  SDLTR-CALLER       PIC X(1)   VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-ASM             VALUE   'A'.
SDL#Y        88 SDLTR-CALLER-COBOL           VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-C               VALUE   'C'.
SDL#Y      05  SDLTR-VAR-TYPE     PIC X(1)   VALUE   SPACE.
SDL#Y        88 SDLTR-VAR-TYPE-ALL           VALUE   ' '.
SDL#Y        88 SDLTR-VAR-TYPE-BIN           VALUE   'B'.
SDL#Y        88 SDLTR-VAR-TYPE-CHR           VALUE   'C'.
SDL#Y        88 SDLTR-VAR-TYPE-DEC           VALUE   'D'.
SDL#Y        88 SDLTR-VAR-TYPE-HEX           VALUE   'X'.
SDL#Y      05  SDLTR-TIMESTAMP    PIC X(1)   VALUE   'L'.
SDL#Y        88 SDLTR-TMSTP-GMT              VALUE   'G'.
SDL#Y        88 SDLTR-TMSTP-LOC              VALUE   'L'.
SDL#Y      05  SDLTR-WRITE-IMM    PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-WRITE-IMM-ON           VALUE    1.
SDL#Y        88 SDLTR-WRITE-IMM-OFF          VALUE    0.
SDL#Y      05  SDLTR-CONS-MSG-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-CONS-MSG-OFF           VALUE    1.
SDL#Y        88 SDLTR-CONS-MSG-ON            VALUE    0.
SDL#Y      05  SDLTR-TIMING-IND   PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TIMING-ON              VALUE    1.
SDL#Y        88 SDLTR-TIMING-OFF             VALUE    0.
SDL#Y      05  SDLTR-TRACECTL     PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TRACECTL-ON            VALUE    1.
SDL#Y        88 SDLTR-TRACECTL-OFF           VALUE    0.
SDL#Y      05  SDLTR-LOG-MODE-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-LOG-MODE-ON            VALUE    1.
SDL#Y        88 SDLTR-LOG-MODE-OFF           VALUE    0.
SDL#Y      05  SDLTR-DUPLICAT-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-DUPLICAT-ON            VALUE    1.
SDL#Y        88 SDLTR-DUPLICAT-OFF           VALUE    0.
SDL#Y      05  SDLTR-SKIP-NAME1   PIC X(8)   VALUE   LOW-VALUE.
```

```
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME2   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME3   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-RESERVED     PIC X(18)  VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SYSTEM-AREA  PIC X(1800) VALUE  LOW-VALUE.
SDL#Y      05  SDLTR-VALIDCHK-Z   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y
SDL#Y  01  SDLTR-WORK  GLOBAL.
SDL#Y      05  SDLTR-LENGTH       PIC S9(9) BINARY.
SDL#Y      05  SDLTR-SAVERC       PIC S9(4) BINARY.
SDL#Y      05  SDLTR-INDEX1       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX2       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX1-NUM   PIC 9(8).
SDL#Y      05  SDLTR-INDEX2-NUM   PIC 9(8).
SDL#Y      05  SDLTR-SET-TRUE     PIC X(4)  VALUE 'TRUE'.
SDL#Y      05  SDLTR-GEN-DATE     PIC X(11) VALUE '15 Mar 2015'.
SDL#Y      05  SDLTRACE           PIC X(8)  VALUE 'SDLTRACE'.
SDL#***------------------------------------------------------------***
       01 Log-Record           pic x(80).

       Linkage Section.
       01 Log-Data             pic x(80).

       Procedure Division using Log-data.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'LOGMOD11' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE 'SDLAPPL1' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 60         TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-ON   TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-OFF  TO TRUE
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE.
SDL#D      .
SDL#***------------------------------------------------------------***
       Main.
           move Log-Data to Log-Record
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'Log-Record' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H      BY CONTENT LENGTH OF Log-Record
SDL#E      END-CALL
SDL#O      MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
```

```
SDL#Z *    MOVE 'STOP '    TO SDLTR-TYPE
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
           goback.

       End program LOGMOD11.
```

The program above now contains all necessary statements for a basic logging function. After making the changes to the JCL (as required by your installation) you can submit the job for compilation. It will then be linked into the load library under the name LOGMOD11. The source is stored under LOGMXD11 to distinguish it from the original code.

The logging module LOGMOD11 can be called from any program with 80 bytes of data to be logged. As an example look at module SAMPLG11 in library *“user-id”.SDLTRACE.DEMO.COBOL:*

```
       ID Division.

       Program-ID  SAMPLG11.

       Data Division.

       Working-Storage Section.

       01  log-data                  pic  x(80) value space.
       01  log-routine               pic  x(8)  value 'LOGMOD11'.
       01  application-1             pic  x(8)  value 'SAMPLG11'.

       Procedure Division.
           move '--- Logging example ---' to log-data
           call log-routine            using log-data
           perform 10 times
             move 'Data to be logged' to log-data
             call log-routine      using log-data
           end-perform
           move '--- End Logging example ---' to log-data
           goback.

       Program-ID  SAMPLG11.
```

The program calls the log routine LOGMOD11, first with a heading line, then several times with identical data, and at last with a terminating line:

When this program is run, a log dataset is allocated using the parameters from the panel that was used to construct the log module. The name of the dataset is listed in the job output and consists of the user-id, the application-id, the jobname, a code representing the Job-ID and the date and time of allocation:

```
user-id.SDLAPPL1.SAMPLG11.ZL.D110315.T1109L
```

The data to be logged is written into this dataset together with a timestamp and some other information.

```
--- Logging example ---
Data to be logged
Data to be logged
Data to be logged
Data to be logged
Data to be logged
Data to be logged
Data to be logged
Data to be logged
Data to be logged
Data to be logged
--- End Logging example ---
```

Each log line is actually 133 bytes long, out of which only the first 80 are shown in the example above because all 133 bytes will not fit on one line in this document. The remaining 53 bytes of each line are listed here separately:

```
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.470008
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509587
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509607
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509621
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509635
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509649
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509664
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509679
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509694
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509708
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.509722
SAMPLG11SAMPLG11E003   1 L 2015-03-15 11:09:42.511903
```

As seen above, all 80-byte log records are displayed just as they were produced by the program.
The information to the right of each log record consist of the following:

First the name of the program that **called** the log-routine is listed. The log-routine determines the caller by stepping back in the call chain to locate the actual name, which may sometimes not be possible. In that case an asterisk “*” is listed to indicate that no valid name could be found. For more details on this feature please see chapter 5: “*Determining the names of calling programs”.*

The caller name is followed by the name of the job which in our case is identical to the name of the caller. Adjacent to the jobname is the name of the system image on which the job is executed. Following that is a four-byte field containing the duplication counter, indicating the number of occurrences of lines with identical data in column 1 through 80. This check for identical data, however, is only performed if the field `Count duplicates` is set to “`Y`” in the ATRACE panel that is used to generate the log routine. Since this parameter was set to “`N`” when LOGMOD11 was created, there is no check for duplicates, each record is shown although they are mostly identical and the duplication counter is 1 for each record.
The ‘L’ which follows is the mode indicator. For trace records it is set to ‘T’, for log records it is set to ‘L’.
The date and time that follows the mode indicator is identical for log and trace records.

For an example with duplicates we will now create a log module using the template LOGMOD12 which is an exact copy of the skeleton LOGMOD11. The resulting log module will be different, however, since we will turn on checking for duplicates. Please go to library.*“user-id”.*SDLTRACE.DEMO.CNTL, step down to member ATRACE and type “**ex**” (short for “exec”) in front of it. The panel being displayed should be modified slightly to look like this:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                        Screen 8 of 6
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         LOGMOD12
Output member        LOGMXD12        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (T/L)
Application-ID       SDLAPPL2        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)   60__ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     Y (Y/N)         Include string #1  LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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
```

Please make sure that the values are indeed as shown above, especially the field `Count duplicates` the value of which should be set to “`Y`”. This will turn on the checking for duplicates when creating log records.

To process this panel, please enter “x” in the top right field and hit the “`Enter`” key. The following messages will be generated for the input values on the panel above:

```
SDLTRACE - Version 4.5.23      15 Mar 2015 11:22:01 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - -----------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD12
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD12
SDLTRACE - -----------------------------------
SDLTRACE - Number of lines in original:      39
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      166
SDLTRACE - -----------------------------------
SDLTRACE - End of process  RC=0
```

Hitting the “`Enter`” key again will display the changed module LOGMXD12 in SPF edit mode. This module should be identical to the one we did have before, except for one line:

```
SDL#K      SET  SDLTR-DUPLICAT-ON   TO TRUE
```

In the previous version of LOGMOD11 this line had been:

```
SDL#K      SET  SDLTR-DUPLICAT-OFF  TO TRUE
```

which caused the checking for duplicates to be suppressed. Please submit the job for compilation and linking under the name LOGMOD12.

The program that shows the effect of duplicate checking is SAMPLG12. It is a copy of SAMPLG11, except that it is calling LOGMOD12 instead of LOGMOD11. When SAMPLG12 is run the following log dataset will be produced:

```
--- Logging example ---
Data to be logged
Data to be logged
--- End Logging example ---
```

with the rightmost 53 columns:

```
                           SAMPLG12SAMPLG12E003   1 L 2015-03-15 11:23:39.674058
                           SAMPLG12SAMPLG12E003   1 L 2015-03-15 11:23:39.707410
                           SAMPLG12SAMPLG12E003   9 L 2015-03-15 11:23:39.707542
                           SAMPLG12SAMPLG12E003   1 L 2015-03-15 11:23:39.709599
```

Line 2 is listed as before. Since line 3 to 11 are identical to line 2, they are displayed in just one line with the duplication counter set to 9 (9 lines equal to the immediately preceding line).

Duplication checking should be used with care, because it might yield unexpected results as the following example will show. Please look at program SAMPLG13 in library “*user-id”.*SDLTRACE.DEMO.COBOL:

```
ID Division.

Program-ID  SAMPLG13.

Data Division.

Working-Storage Section.
01  log-data                  pic  x(80) value space.
01  log-routine               pic  x(8)  value 'LOGMOD12'.

Procedure Division.
    move '--- Logging example ---' to log-data
    call log-routine            using log-data
    perform 10 times
      move 'Data to be logged' to log-data
      call log-routine      using log-data
    end-perform
    goback.

End Program SAMPLG13.
```

The program is similar to SAMPLG12 except that the terminating line is missing. When this program is run, the following log records are being created:

```
--- Logging example ---
Data to be logged
```

with the 53 rightmost columns:

```
                           SAMPLG13SAMPLG13E003   1 L 2015-03-15 11:28:26.216808
                           SAMPLG13SAMPLG13E003   2 L 2015-03-15 11:28:26.249052
```

Why are there only two log lines with only the first one of the ten duplicate lines listed? The explanation is simple: When the log routine detects the first equal line, it begins to count without writing anything to the log file since the next call might again be an equal line, and this continues until a call with an unequal line forces the accumulated data to be written to the log dataset. However, in this case this terminating call never comes because the caller simply stops without giving the log routine a chance to clear its buffers.

Usually log records are not identical and for simple logging applications, where the length of one record is just 80 bytes, using the check for duplicates is not recommended. The use of duplicate checking is very useful, however, when the length of a log item is larger than 80 and the information to be recorded is variable, with many records that are smaller than the maximum allowed. For more information please see the chapter: *”Logging large data items”.*

The timestamp is formatted by the trace engine which reads the system clock immediately before storing the generated log record into the trace buffer. Thus for every log record the time of its creation is stored, which can be used to compute elapsed times between arbitrary events. In the chapter *“Measuring execution times”* this will be explained in more detail.

The examples above show the basic logging function. There are many additional options which can be specified in the main panel ATRACE and whose use is explained in the following chapters.

Before doing that, however, here is a brief look at the top line. It does not specify a function or option, it is rather used to control the panel itself:

```
Run Jcl or eXec now or Delete or display Next/Previous screen: _ (J X D N P)
                                                                  (or: I/R/E)
```

There are five letters that may be entered in the input field above: J, X, D, N and P. (Lower case entries are automatically translated to upper case). J and X are used to have the specified parameters in the panel processed by the COBOL SDLTRACE pre-processor “SDLPREP”, a REXX program that scans the module to be prepared for logging. There are two modes of operation for this pre-processor: batch and TSO. In batch mode a job is prepared that can be submitted for execution. In TSO mode the pre-processor is called directly from the panel. Selection between the two modes is made by specifying “**j**” or “**x**” respectively. In the examples we will always use “**x**” followed by the “`Enter`” key to submit the panel for execution in TSO. Of course you could also use “**j**” and generate the programs in batch mode if you prefer to run REXX procedures that way.

There may be up to nine copies of the panel with different parameters that are stored in the user’s ISPF profile dataset, and the letters “**n**” and “**p**” may be used to switch between them. A panel that is not needed anymore can be deleted by specifying “**d**”.

A panel does not have an explicit name. Therefore the panels are distinguished by a combination of the four values:

```
        Input dataset
        Output dataset
        Input member
        Output member
```

A new panel is generated automatically if any of the four values above is changed and if less than nine panels exist in the users ISPF profile. If nine panels are already defined, then a change of any of the four values above will replace the current panel upon processing.

The additional values (`I/R/E)` may also be used. They are a combination of the edit options `Insert/Remove/Edit` and the action “x”.

In the following chapters some of the features of SDLTRACE are explained in more detail.

Chapter 2. Controlling the allocation of log datasets

There are four parameters to control the allocation of log datasets: DSN qualifier, Application-ID, space and time. These parameters are specified on the ATRACE panel in the section:

```
DSN qualifier        user-id_         Trace/Log mode      L (T/L)
Application-ID       SDLAPPL2         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)   60__ (0-1440)    Trace variables     Y (Y/N)
```

The `DSN qualifier` specifies the high level of the log dataset that is automatically allocated when the first call to the log routine is issued. It may be any name which is a valid DSN and for which the job has the authorization (in RACF or similar systems) for allocation. Within all sample jobs we use the current user's ID which of course does have this authorization when the jobs are submitted by TSO. If the program with logging is to be executed in another environment, for example in CICS, IMS, WLM, etc., then the high level should be one that is authorized for "Write Access" in those systems.

The `DSN qualifier` is not limited to just one level; it could also be `ABC.LOG`, for example (if allocation authority for `ABC.LOG` exists). In case that this authorization is missing, it will not be possible to allocate the log dataset and an error message will be issued. No log data will be produced in this case; the program, however, will be executed as if no log code were present.

The `Application-ID` determines the second level of the log dataset name. It may be any valid DSN level and is used to distinguish between different programs or a set of programs belonging to a group. If in one job step several programs with log code are executed and the application-ID is the same, then the log data is recorded in the single dataset with that particular application-ID. This enables the user to group log data according to individual requirements.

In the first log routine used above we did have the application-ID "SDLAPPL1", which was permanently compiled into the routine. If another application-ID were required we could of course generate another log routine with a new application-ID. There is, however, a better way to control application-IDs, and that is to make them variable, so that the calling program actually supplies it. For an example please look at module LOGMOD21 in library *"user-id".*SDLTRACE.DEMO.COBOL:

```
Identification Division.
Program-ID. LOGMOD21.

Data Division.
Working-storage Section.
01 Log-Record           pic x(80).

Linkage Section.
01 Log-Data             pic x(80).
01 Log-Appl             pic x(8).

Procedure Division using Log-Data Log-Appl.
Main.
    move Log-Appl to SDLTR-APPL-ID
    move Log-Data to Log-Record
    goback.

End program LOGMOD21.
```

When this program is compiled the way it is then an error is generated because the variable `SDLTR-APPL-ID` is not defined. This is normal since the program is not yet complete, but rather only a skeleton for generation of the "real" LOGMOD21 by the Cobol SDLTRACE pre-processor.

Just as for example LOGMOD11, please call up the panel ATRACE again by going to the library *"user-id"*.SDLTRACE.DEMO.CNTL, step down to member ATRACE and type "**ex**" in front of it. Then please set the values according to the listing shown here:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                             Screen 6 of 6
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         LOGMOD21
Output member        LOGMXD21        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (T/L)
Application-ID       ________        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)   60__ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     N (Y/N)         Include string #1  LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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
```

The application-ID is left blank because we will supply it later with each individual logging call. Also please make sure that the value of `Count duplicates` is set to "`N`". When this panel is processed by entering "x" in the top right corner and hitting the "`Enter`" key we will get the following messages:

```
SDLTRACE - Version 4.5.23      15 Mar 2015 12:47:05 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - ------------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD21
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD21
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      40
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      169
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

Now hit "`Enter`", and the completed version of LOGMOD21 will be displayed:

```
       Identification Division.
       Program-ID. LOGMOD21.

       Data Division.
       Working-Storage Section.
```

```
SDL#***------------------------------------------------------------***
SDL#Y  01  SDLTR-PARM  GLOBAL.
SDL#Y      05  SDLTR-VALIDCHK-A   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VERSION      PIC X(8)   VALUE  'VER 4.5 '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-DSN-HILEVEL  PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-APPL-ID      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PGMNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-JOBNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TYPE         PIC X(5)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TEXT         PIC X(50)  VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-START   PIC X(8)   VALUE  'PERFORM '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-END     PIC X(8)   VALUE  '--END-- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-LABEL        PIC X(8)   VALUE  '------- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PRI-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-SEC-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-VAR-LENGTH   PIC 9(4)   VALUE 31   BINARY.
SDL#Y      05  SDLTR-FLD-LENGTH   PIC 9(4)   VALUE 13   BINARY.
SDL#Y      05  SDLTR-THRESHOLD    PIC 9(4)   VALUE 0    BINARY.
SDL#Y      05  SDLTR-NEWTIM       PIC 9(4)   VALUE 1440 BINARY.
SDL#Y      05  SDLTR-RETN-CODE    PIC 9(2)   VALUE   0.
SDL#Y      05  SDLTR-RETN-CBIN    PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TRACE        PIC 9(1)   VALUE   1.
SDL#Y        88 SDLTR-TRACE-ON               VALUE   1.
SDL#Y        88 SDLTR-TRACE-OFF              VALUE   0.
SDL#Y      05  SDLTR-CALLER       PIC X(1)   VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-ASM             VALUE   'A'.
SDL#Y        88 SDLTR-CALLER-COBOL           VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-C               VALUE   'C'.
SDL#Y      05  SDLTR-VAR-TYPE     PIC X(1)   VALUE   SPACE.
SDL#Y        88 SDLTR-VAR-TYPE-ALL           VALUE   ' '.
SDL#Y        88 SDLTR-VAR-TYPE-BIN           VALUE   'B'.
SDL#Y        88 SDLTR-VAR-TYPE-CHR           VALUE   'C'.
SDL#Y        88 SDLTR-VAR-TYPE-DEC           VALUE   'D'.
SDL#Y        88 SDLTR-VAR-TYPE-HEX           VALUE   'X'.
SDL#Y      05  SDLTR-TIMESTAMP    PIC X(1)   VALUE   'L'.
SDL#Y        88 SDLTR-TMSTP-GMT              VALUE   'G'.
SDL#Y        88 SDLTR-TMSTP-LOC              VALUE   'L'.
SDL#Y      05  SDLTR-WRITE-IMM    PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-WRITE-IMM-ON           VALUE    1.
SDL#Y        88 SDLTR-WRITE-IMM-OFF          VALUE    0.
SDL#Y      05  SDLTR-CONS-MSG-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-CONS-MSG-OFF           VALUE    1.
SDL#Y        88 SDLTR-CONS-MSG-ON            VALUE    0.
SDL#Y      05  SDLTR-TIMING-IND   PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TIMING-ON              VALUE    1.
SDL#Y        88 SDLTR-TIMING-OFF             VALUE    0.
SDL#Y      05  SDLTR-TRACECTL     PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TRACECTL-ON            VALUE    1.
SDL#Y        88 SDLTR-TRACECTL-OFF           VALUE    0.
SDL#Y      05  SDLTR-LOG-MODE-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-LOG-MODE-ON            VALUE    1.
SDL#Y        88 SDLTR-LOG-MODE-OFF           VALUE    0.
SDL#Y      05  SDLTR-DUPLICAT-IND PIC 9(1)   VALUE    0.
```

```
SDL#Y        88 SDLTR-DUPLICAT-ON           VALUE    1.
SDL#Y        88 SDLTR-DUPLICAT-OFF          VALUE    0.
SDL#Y      05  SDLTR-SKIP-NAME1   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME2   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME3   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-RESERVED     PIC X(18)  VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SYSTEM-AREA  PIC X(1800) VALUE  LOW-VALUE.
SDL#Y      05  SDLTR-VALIDCHK-Z   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y
SDL#Y  01  SDLTR-WORK  GLOBAL.
SDL#Y      05  SDLTR-LENGTH       PIC S9(9) BINARY.
SDL#Y      05  SDLTR-SAVERC       PIC S9(4) BINARY.
SDL#Y      05  SDLTR-INDEX1       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX2       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX1-NUM   PIC 9(8).
SDL#Y      05  SDLTR-INDEX2-NUM   PIC 9(8).
SDL#Y      05  SDLTR-SET-TRUE     PIC X(4)  VALUE 'TRUE'.
SDL#Y      05  SDLTR-GEN-DATE     PIC X(11) VALUE '20 Jul 2015'.
SDL#Y      05  SDLTRACE           PIC X(8)  VALUE 'SDLTRACE'.
SDL#***------------------------------------------------------------***
       01 Log-Record            pic x(80).

       Linkage Section.
       01 Log-Data              pic x(80).
       01 Log-Appl              pic x(8).

       Procedure Division using Log-Data Log-Appl.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'LOGMOD21' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE '        ' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 60         TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-ON   TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-OFF  TO TRUE
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE.
SDL#D      .
SDL#***------------------------------------------------------------***
       Main.
           move Log-Appl to SDLTR-APPL-ID
           move Log-Data to Log-Record
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'Log-Record' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H      BY CONTENT LENGTH OF Log-Record
```

```
SDL#E      END-CALL
SDL#O      MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#Z *    MOVE 'STOP '    TO SDLTR-TYPE
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
           goback.
       End program LOGMOD21.
```

This completed version of LOGMOD21 is saved under the name of LOGMXD21 to distinguish it from the skeleton LOGMOD21. Before hitting PF3 however, please submit the job for compilation and linkage into the load library. The load module is called LOGMOD21 and is now ready for use.

An example for the use of LOGMOD21 can be found in *"user-id"*.SDLTRACE.DEMO.COBOL in program SAMPLG21:

```
ID Division.

Program-ID  SAMPLG21.

Data Division.

Working-Storage Section.
01  log-data                  pic  x(80).
01  log-routine               pic  x(8)  value 'LOGMOD21'.
01  app-name                  pic  x(8).

Procedure Division.
    move 'LOG-DATA for Application 2X' to log-data
    move 'SDLAPP2X'     to app-name
    call log-routine using log-data app-name

    move 'LOG-DATA for Application 2Y' to log-data
    move 'SDLAPP2Y'     to app-name
    call log-routine using log-data app-name
    perform 5 times
      move 'More LOG-DATA for Application 2X' to log-data
      move 'SDLAPP2X'    to app-name
      call log-routine using log-data app-name

      move 'More LOG-DATA for Application 2Y' to log-data
      move 'SDLAPP2Y'     to app-name
      call log-routine using log-data app-name
    end-perform

    goback.

End Program SAMPLG21.
```

When this program is executed, two log datasets are being created and their names listed in the Joblog:

```
+SDLTRACE - user-id.SDLAPP2X.SAMPLG21.A3.D150315.T1258L
+SDLTRACE - user-id.SDLAPP2Y.SAMPLG21.A3.D150315.T1258L
```

These two datasets will contain the following information. In the first one we have:

```
LOG-DATA for Application 2
More LOG-DATA for Application 2X
More LOG-DATA for Application 2X
More LOG-DATA for Application 2X
More LOG-DATA for Application 2X
More LOG-DATA for Application 2X

                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.649085
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728177
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728219
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728258
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728298
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728337
```

and in the second one:

```
LOG-DATA for Application 3
More LOG-DATA for Application 2Y
More LOG-DATA for Application 2Y
More LOG-DATA for Application 2Y
More LOG-DATA for Application 2Y
More LOG-DATA for Application 2Y

                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.693061
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728200
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728238
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728278
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728318
                             SAMPLG21SAMPLG21E003   1 L 2015-03-15 12:58:36.728358
```

As can be seen in program SAMPLG21, there may be many different application names supplied to LOGMOD21, so that the number of individual log datasets is actually not limited. An application can thus have as many different log streams as it needs.

For another example please look at program SAMPLH21:

```
       ID Division.

       Program-ID  SAMPLH21.

       Data Division.

       Working-Storage Section.

       01  log-data.
         05  log-rec             pic  x(13) value 'Log record # '.
         05  log-count           pic  9(2)  value zero.
         05  log-fill            pic  x(65) value space.
       01  log-routine           pic  x(8)  value 'LOGMOD21'.
       01  app-name              pic  x(8)  value 'SDLAPP00'.

       Procedure Division.

           perform varying log-count from 1 by 1 until log-count > 10
             move log-count to app-name(7:2)
             call log-routine  using  log-data  app-name
           end-perform
           goback.

       End Program SAMPLH21.
```

This is an example of dynamically creating application names and then logging data in the corresponding log datasets. Just run the program; ten files will be allocated and each one will contain a single log record.

In addition to the first two parts of the log dataset name (high level index and application-ID), there is a third part which is automatically inserted, and that is the name of the job or monitor under which the program being logged is run. Following the jobname is a two-character code representing the Job-ID.

The last two parts of a log dataset name are the date and the time of its allocation in the following format: "Dyymmdd.ThhmmL". The last letter of the time part is set to "L" to distinguish a log file from trace files where the last letter may be "A", "B" or "C".

This completes the building of a log dataset name, and the remaining two factors of allocation that the user can control are space and time (duration during which a log dataset is active). Before going into detail about that we will look briefly at another related matter: Job or Monitor control.

The parameter `JOB-ID check` is used to limit the actual logging performed by a module in an application to just the job or all jobs starting with a certain string. An asterisk '*' in this field means that a log should be generated irrespective of the job or monitor name under which it is executed. If a name is specified then a log is produced only if the actual job or monitor is equal to that name. If the first part of a name with an asterisk appended (as in 'ABC*') is specified then a log is generated only for jobs or monitors whose name starts with that string. For all other jobs no log is produced (i.e. no log dataset is allocated) and the application runs as if no log code existed, except that the return-code (Register 15) passed back from SDLTRACE is set to 4.

To preserve an application's own return-code, just set the option `Save RETURN-CODE` to '`Y`'. This option actually means that an application's return-code register should be preserved across the call to the log routine. Since the invocation of SDLTRACE is an external call, 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 is therefore the recommended setting. Only in very special cases where the user is aware of the possible side-affects and needs to know if a log record really was written should this value be set to '`N`'.

Instead of a single jobname it is also possible to provide a list of names for which logging is to be enabled. This list is specified in library *"user-id".*SDLTRACE.DEMO.ASM in the Assembler module SDLJOBLG. For information regarding its use please see the comments in that module. To enable the use of the list the program SDLJOBLG must be linked into SDLTRACE and the option `JOB-ID check` must be set to the string "`JOB-LIST`" in the ATRACE panel.

Now we look at the remaining two parameters which enable the user to control the allocation and the life span of the log datasets.

The space parameter `DSN alloc (tracks)` specifies the number of tracks between 1 and 9999 to be allocated when a log dataset is required. A new 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 time part reflects the new time. For an example please look at program SAMPLI21 in library *"user-id".*SDLTRACE.DEMO.COBOL:

```
ID Division.

Program-ID  SAMPLI21.

Data Division.

Working-Storage Section.
01  log-data                 pic  x(80).
01  log-routine              pic  x(8)  value 'LOGMOD21'.
01  app-name                 pic  x(8)  value 'SDLAPP21'.
01  cnt                      pic  9(4)  value zero.
Procedure Division.
    perform varying cnt from 1 by 1 until cnt > 1000
      move 'Multiple file test, record 0000.' to log-data
```

```
             move cnt to log-data(28:4)
             call log-routine using log-data app-name
           end-perform

           goback.

       End Program SAMPLI21.
```

When this program is run a single log file will be allocated which will then be filled with 1000 log records. This is because the log routine LOGMOD21 was generated with a primary allocation of 100 tracks.

To test the allocation of secondary files we will now create LOGMOD22 so that just one track will be allocated by specifying that in the ATRACE panel:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                                 Screen 6 of 6
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         LOGMOD22
Output member        LOGMXD22        Insert/Remove/Edit I (I/R/E)

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

Count duplicates     N (Y/N)         Include string #1  LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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
```

Please make sure that `DSN alloc (tracks)` is set to `1` to have just one track allocated for the log dataset. When this panel is processed by entering “x” in the top right corner and hitting the “`Enter`” key we will get the following messages:

```
SDLTRACE - Version 4.5.23      16 Mar 2015 08:19:22 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - -----------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD22
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD22
SDLTRACE - -----------------------------------
SDLTRACE - Number of lines in original:      40
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      169
SDLTRACE - -----------------------------------
SDLTRACE - End of process  RC=0
***
```

Now hit "`Enter`" , and the completed version of LOGMOD22 will be displayed. The only difference compared to LOGMOD21 is in the following lines:

```
SDL#K      MOVE 1          TO SDLTR-PRI-TRKS
SDL#K      MOVE 1          TO SDLTR-SEC-TRKS
```

which had been before:

```
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
```

Please submit this job for compilation and linking so that we now have a log module that allocates just one track for a log dataset. When SAMPLG22 is run the system will allocate another log dataset after 392 records since that many will fit on one track on a 3390 disk device.

When we look at the job output, however, in almost all cases we will see only one log dataset. The reason for this is the structure of the DSN, which is formed using application name, jobname, job-ID and the timestamp. The second DSN is therefore identical to the first one, except when the minute changes. Since for just 1000 records the execution time is less than one second it is highly unlikely that there will be a change in the minute part of the DSN. As a result the data will be appended to the previous log records. (The re-allocation of the datasets after 392 records can be seen in the job JCL SMS messages).

The usual allocation size for log datasets is between 100 and 500 tracks, which allows between 39.200 and 196.000 log records to be written before a new dataset is allocated. If there is a need for larger datasets they may of course also be specified. The largest possible allocation (9999 tracks) will allow 3.919.608 records to be stored in a single dataset before a new one is needed. In order to avoid problems when trying to extend a dataset on the initial volume the allocation parameter "multi-volume" is set to 15.

The last parameter that is used to control allocation of log datasets is time, that is the duration during which a dataset should be active, specified on the ATRACE panel in the variable `DSN time (minutes)`. Its value is specified in minutes and can be between 1 and 1440 with the restriction that it must be divisible into 1440 without remainder. This means that there should be a whole number of equal periods during one day.

To show the operation of the time parameter we will set the value to 1 and then run the program SAMPLG23 which has delays built in so that it runs several minutes and forces the allocation of several datasets because the active period for a log expires.

Now please call up ATRACE again and set its values according to the following list:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                              Screen 6 of 6
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         LOGMOD23
Output member        LOGMXD23       Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id        Trace/Log mode     L (T/L)
Application-ID       ________       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)   1___ (0-1440)  Trace variables    Y (Y/N)

Count duplicates     N (Y/N)        Include string #1  LOG-RECORD___________
Console messages     Y (Y/N)        Include string #2  _____________________
Save RETURN-CODE     Y (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
```

Please make sure that `DSN time (minutes)` is set to `1` to force a new log dataset to be allocated every minute. When this panel is processed by entering “x” in the top right corner and hitting the “`Enter`” key we will get the following messages:

```
SDLTRACE - Version 4.5.23      16 Mar 2015 08:32:18 USER-ID
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - ------------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD23
SDLTRACE - Output library:    uder-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD23
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      40
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      169
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

Now hit “`Enter`” , and the completed version of LOGMOD23 will be displayed. The only change to LOGMOD21 is in the following line:

```
SDL#K      MOVE 1          TO SDLTR-NEWTIM
```

which sets the time parameter to one minute. Now submit this job for compilation and linking, so that we have a log module that allocates a new log dataset every minute. The program SAMPLG23 in library *“user-id”*.SDLTRACE.DEMO.COBOL is used to test this:

```
       ID Division.

       Program-ID  SAMPLG23.

       Data Division.

       Working-Storage Section.

       01  log-data.
         05  log-rec             pic  x(13) value 'Log record # '.
         05  log-count           pic  9(5)  value zero.
         05  log-fill            pic  x(62) value space.
       01  log-routine           pic  x(8)  value 'LOGMOD23'.
       01  app-name              pic  x(8)  value 'SDLAPP23'.

       01  wait                  pic  x(8)  value  'SDLWAIT'.
       01  wait-parm.
         05                      pic s9(4)  comp  value 8.
         05  wait-time.
           10  wait-time-hh      pic  9(2)  value  zero.
           10  wait-time-mm      pic  9(2)  value  zero.
           10  wait-time-ss      pic  9(2)  value  1.
           10  wait-time-th       pic  9(2)  value  zero.

       Procedure Division.

           perform varying log-count from 1 by 1 until log-count > 180
             call log-routine  using  log-data  app-name
             call wait         using  wait-parm
           end-perform
           goback.

       end program SAMPLG23.
```

This program writes 180 log records, and after each of the writes it waits for one second before writing the next one. This is done by calling SDLWAIT, a special module which is used to introduce waits in COBOL programs. The source code is stored in library *“user-id”*.SDLTRACE.DEMO.ASM*,* the load module is linked into “*user-id”*.SDLTRACE.DEMO.LOAD so that it can be used by any of the sample programs.

When SAMPLG23 is run it allocates the first log dataset and records data until the minute value of the system clock changes. Then the current dataset is closed and a new one is allocated. This continues until 180 log records are written. The log datasets produced look like this:

```
User-id.SDLAPP23.SAMPLG23.FT.D150316.T0845L
User-id.SDLAPP23.SAMPLG23.FT.D150316.T0846L
User-id.SDLAPP23.SAMPLG23.FT.D150316.T0847L
User-id.SDLAPP23.SAMPLG23.FT.D150316.T0848L
```

The first dataset contains the following records:

```
Log record # 00001
Log record # 00002
Log record # 00003
Log record # 00004
Log record # 00005
```

and the rightmost 53 columns are:

```
                           SAMPLG23SAMPLG23E003   1 L 2015-03-16 08:45:55.756583
                           SAMPLG23SAMPLG23E003   1 L 2015-03-16 08:45:56.791594
                           SAMPLG23SAMPLG23E003   1 L 2015-03-16 08:45:57.791690
                           SAMPLG23SAMPLG23E003   1 L 2015-03-16 08:45:58.791751
                           SAMPLG23SAMPLG23E003   1 L 2015-03-16 08:45:59.791893
```

In this particular example the job started at `08:45:55`. Therefore only 5 records are contained in the first dataset, because after five seconds the minute changes form 55 to 56 and thus a new dataset is allocated. The other 175 records are in the remaining datasets. Their contents is similar to that of the first dataset and it is therefore not necessary to list them here.

For most applications the parameter `DSN time (minutes)` is set to `60` , so that a new dataset is allocated at least every hour. A value of zero indicates that no new dataset is to be allocated during an entire 24-hour day; it is identical to a specification of 1440 (the number of minutes per day).

Irrespective of the time setting a new dataset is always allocated when the day changes. This is to ensure that data of a particular day is always recorded in a dataset whose name has that day as part of its name:

```
user-id.SDLAPP23.SAMPLG23.AF.D150316.T0847L
```

The dataset above will contain only data generated on March 16, 2015.

When `DSN time (minutes)` is set to `1` a new dataset is allocated every minute. This is useful for online applications under test since during logging the active log dataset is kept open even though the transaction that is calling the log function has terminated. Thus for example under CICS one can inspect the contents of a log dataset only after the log module has closed the logfile. For more details regarding test in online environments and especially the parameter `Enable CICS test` please see the special chapter: **Test in Online Environments-**

Similar to the examples in chapter 1, the generation of the modules described above may be performed in batch mode automatically, without having to enter any data into the ATRACE panel manually. Please display the relevant members by specifying on the ISPF DSN display:

```
e        user-id.SDLTRACE.DEMO.CNTL(SDLPRE2*)
```

The following members will be shown:

```
_________ SDLPRE21
_________ SDLPRE22
_________ SDLPRE23
```

Just submit `SDLPRE21`, `SDLPRE22` and `SDLPRE23` by entering “j” on the command line. This will use the ATRACE panel parameters defined in `SDLPRC21`, `SDLPRC22` and `SDLPRC23` to generate the logging modules `LOGMOD21`, `LOGMOD22` and `LOGMOD23`.

The sample jobs in the COBOL library which use these modules will be displayed when specifying:

```
e        user-id.SDLTRACE.DEMO.COBOL(sampl*21)
```

on the ISPF DSN display. The following members should be shown:

```
_________ SAMPLG21
_________ SAMPLH21
_________ SAMPLI21
```

These jobs can now be run to generate the example log datasets.

Similarly the jobs `SAMPLG22` and `SAMPLG23` can be submitted to produce the output of the last two examples.

So far we have always had data records with a size of 80 bytes. In the next chapter we will build a log module which accepts a data record with a size of up to 32000 bytes.

Chapter 3. Logging large data items

The standard size of one log record is 80 bytes. For applications which have larger records, a logging module which can handle that can easily be created. The only requirement is that the size of a single log record must be an exact multiple of 80. Thus 160, 240, 800, 3200, etc. can be defined as maximum size of a log item supplied by the application, and a corresponding logging module will then be built. For an example please look at module LOGMOD31 in library "*user-id"*.SDLTRACE.DEMO.COBOL;

```
Identification Division.
Program-ID. LOGMOD31.

Data Division.
Working-Storage Section.
01 Log-Record             pic x(32000).

Linkage Section.
01 Log-Data               pic x(32000).

Procedure Division using Log-Data.
Main.
    move Log-Data to Log-Record
    goback.

End Program LOGMOD31.
```

Just as before, this simple program serves as a base for building the actual logging module LOGMOD31 which will be able to process log items of up to 32000 bytes length. Please call up ATRACE again and set the values according to the following values:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                                 Screen 6 of 6
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         LOGMOD31
Output member        LOGMXD31        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (T/L)
Application-ID       SDLAPPL3        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)   0___ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     Y (Y/N)         Include string #1  LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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
```

Note that in this case the count for duplicates is enabled so that records with little actual data and mostly empty space are checked and only the necessary data is recorded in the log dataset. Also make sure that the field "`Application-ID`" is set to "`SDLAPPL3`" since the sample programs for large log records do not provide an apllication name. Please process this panel by entering "**x**" in the top right entry field and hit "`Enter`". The following messages will be displayed:

```
SDLTRACE - Version 4.5.23      16 Mar 2015 09:03:45 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - ------------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD31
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD31
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      38
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      165
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

Now hit enter to get the complete LOGMOD31 module:

```
       Identification Division.
       Program-ID. LOGMOD31.

       Data Division.
       Working-Storage Section.
SDL#***------------------------------------------------------------***
SDL#Y  01  SDLTR-PARM  GLOBAL.
SDL#Y      05  SDLTR-VALIDCHK-A   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VERSION      PIC X(8)   VALUE  'VER 4.5 '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-DSN-HILEVEL  PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-APPL-ID      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PGMNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-JOBNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TYPE         PIC X(5)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TEXT         PIC X(50)  VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-START   PIC X(8)   VALUE  'PERFORM '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-END     PIC X(8)   VALUE  '--END-- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-LABEL        PIC X(8)   VALUE  '------- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PRI-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-SEC-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-VAR-LENGTH   PIC 9(4)   VALUE 31   BINARY.
SDL#Y      05  SDLTR-FLD-LENGTH   PIC 9(4)   VALUE 13   BINARY.
SDL#Y      05  SDLTR-THRESHOLD    PIC 9(4)   VALUE 0    BINARY.
SDL#Y      05  SDLTR-NEWTIM       PIC 9(4)   VALUE 1440 BINARY.
SDL#Y      05  SDLTR-RETN-CODE    PIC 9(2)   VALUE   0.
SDL#Y      05  SDLTR-RETN-CBIN    PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TRACE        PIC 9(1)   VALUE   1.
SDL#Y        88 SDLTR-TRACE-ON               VALUE   1.
SDL#Y        88 SDLTR-TRACE-OFF              VALUE   0.
SDL#Y      05  SDLTR-CALLER       PIC X(1)   VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-ASM             VALUE   'A'.
```

```
SDL#Y        88 SDLTR-CALLER-COBOL           VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-C               VALUE   'C'.
SDL#Y      05  SDLTR-VAR-TYPE     PIC X(1)   VALUE   SPACE.
SDL#Y        88 SDLTR-VAR-TYPE-ALL           VALUE   ' '.
SDL#Y        88 SDLTR-VAR-TYPE-BIN           VALUE   'B'.
SDL#Y        88 SDLTR-VAR-TYPE-CHR           VALUE   'C'.
SDL#Y        88 SDLTR-VAR-TYPE-DEC           VALUE   'D'.
SDL#Y        88 SDLTR-VAR-TYPE-HEX           VALUE   'X'.
SDL#Y      05  SDLTR-TIMESTAMP    PIC X(1)   VALUE   'L'.
SDL#Y        88 SDLTR-TMSTP-GMT              VALUE   'G'.
SDL#Y        88 SDLTR-TMSTP-LOC              VALUE   'L'.
SDL#Y      05  SDLTR-WRITE-IMM    PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-WRITE-IMM-ON           VALUE    1.
SDL#Y        88 SDLTR-WRITE-IMM-OFF          VALUE    0.
SDL#Y      05  SDLTR-CONS-MSG-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-CONS-MSG-OFF           VALUE    1.
SDL#Y        88 SDLTR-CONS-MSG-ON            VALUE    0.
SDL#Y      05  SDLTR-TIMING-IND   PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TIMING-ON              VALUE    1.
SDL#Y        88 SDLTR-TIMING-OFF             VALUE    0.
SDL#Y      05  SDLTR-TRACECTL     PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TRACECTL-ON            VALUE    1.
SDL#Y        88 SDLTR-TRACECTL-OFF           VALUE    0.
SDL#Y      05  SDLTR-LOG-MODE-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-LOG-MODE-ON            VALUE    1.
SDL#Y        88 SDLTR-LOG-MODE-OFF           VALUE    0.
SDL#Y      05  SDLTR-DUPLICAT-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-DUPLICAT-ON            VALUE    1.
SDL#Y        88 SDLTR-DUPLICAT-OFF           VALUE    0.
SDL#Y      05  SDLTR-SKIP-NAME1   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME2   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME3   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-RESERVED     PIC X(18)  VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SYSTEM-AREA  PIC X(1800) VALUE  LOW-VALUE.
SDL#Y      05  SDLTR-VALIDCHK-Z   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y
SDL#Y  01  SDLTR-WORK  GLOBAL.
SDL#Y      05  SDLTR-LENGTH       PIC S9(9) BINARY.
SDL#Y      05  SDLTR-SAVERC       PIC S9(4) BINARY.
SDL#Y      05  SDLTR-INDEX1       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX2       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX1-NUM   PIC 9(8).
SDL#Y      05  SDLTR-INDEX2-NUM   PIC 9(8).
SDL#Y      05  SDLTR-SET-TRUE     PIC X(4)  VALUE 'TRUE'.
SDL#Y      05  SDLTR-GEN-DATE     PIC X(11) VALUE '28 Jul 2015'.
SDL#Y      05  SDLTRACE           PIC X(8)  VALUE 'SDLTRACE'.
SDL#***------------------------------------------------------------***
       01 Log-Record           pic x(32000).

       Linkage Section.
       01 Log-Data             pic x(32000).

       Procedure Division using Log-Data.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'LOGMOD31' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE 'SDLAPPL3' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
```

```
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 0          TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-ON   TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-ON   TO TRUE
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE.
SDL#D      .
SDL#***------------------------------------------------------------***
      Main.
           move Log-Data to Log-Record
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'Log-Record' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H      BY CONTENT LENGTH OF Log-Record
SDL#E      END-CALL
SDL#O      MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#Z *    MOVE 'STOP '    TO SDLTR-TYPE
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
           goback.

       End program LOGMOD31.
```

The main difference to the previous log modules is the definition of a 32000-byte log record. This will cause SDLTRACE to split the data supplied by the caller into 400 separate 80-byte data records which will then be stored in the standard log dataset. Please submit the job above to compile and link it into the library "*user-id"*.SDLTRACE.DEMO.LOAD.

To test LOGMOD31 the program SAMPLG31 in library "*user-id"*.SDLTRACE.DEMO.COBOL has been provided:

```
       ID Division.
       Program-ID  SAMPLG31.

       Data Division.

       Working-Storage Section.
       01  log-data                  pic  x(32000) value space.
       01  log-count                 pic  9(5)  value zero.
       01  log-routine               pic  x(8)  value 'LOGMOD31'.

       Procedure Division.
           move '--- Logging example for large records ---' to log-data
           call log-routine using log-data
           move 'Record number 00000. The data to be logged may be of an
      -         'y size, as long as it is an exact multiple of 80. For t
      -         'his example we set the size to 32000.'
                                             to log-data
           perform varying log-count from 1 by 1 until log-count > 2
```

```
             move log-count      to log-data(15:5)
             call log-routine using log-data
           end-perform
           move '--- Last record, test complete ---' to log-data
           call log-routine using log-data
           goback.

       End Program SAMPLG31.
```

When this program is run it will generate a log dataset with the following lines:

```
-- Logging example for large records --



Record number 00001. The data to be logged may be of any size, as long as it is
an exact multiple of 80. For this example we set the size to 32000.



Record number 00002. The data to be logged may be of any size, as long as it is
an exact multiple of 80. For this example we set the size to 32000.



-- Last record, test complete --
```

with the rightmost 53 columns:

```
                           SAMPLG31SAMPLG31E003   1,L 2015-03-16 09:36:14.706580
                           SAMPLG31     1         2.L 2015-03-16 09:36:14.706580
                           SAMPLG31   397       399.L 2015-03-16 09:36:14.706580
                           SAMPLG31     1       400;L 2015-03-16 09:36:14.706580
                           SAMPLG31SAMPLG31E003   1,L 2015-03-16 09:36:14.739160
                           SAMPLG31     1         2.L 2015-03-16 09:36:14.739160
                           SAMPLG31     1         3.L 2015-03-16 09:36:14.739160
                           SAMPLG31   396       399.L 2015-03-16 09:36:14.739160
                           SAMPLG31     1       400;L 2015-03-16 09:36:14.739160
                           SAMPLG31SAMPLG31E003   1,L 2015-03-16 09:36:14.742044
                           SAMPLG31     1         2.L 2015-03-16 09:36:14.742044
                           SAMPLG31     1         3.L 2015-03-16 09:36:14.742044
                           SAMPLG31   396       399.L 2015-03-16 09:36:14.742044
                           SAMPLG31     1       400;L 2015-03-16 09:36:14.742044
                           SAMPLG31SAMPLG31E003   1,L 2015-03-16 09:36:14.744993
                           SAMPLG31     1         2.L 2015-03-16 09:36:14.744993
                           SAMPLG31   397       399.L 2015-03-16 09:36:14.744993
                           SAMPLG31     1       400;L 2015-03-16 09:36:14.744993
```

Large log records are divided up into parts of 80 bytes each, formatted just as short log records except for some differences in the rightmost 52 bytes. In order to distinguish continuation records from single entries the following markers are set immediately to the left of the log marker “L”. A comma “,“ denotes the first part of a long record, a decimal point “.“ denotes all subsequent parts, and a semicolon “;“ is set to signify the last part. The timestamp for all parts of a record is of course identical; it is the time when the application called the logging system to store the large record.

Just as in short records the first part of a large record contains the calling program, the job or monitor name and the system image where the job is run. The number to the left of the “L” is the part number which for the first part is always 1. For all subsequent parts the jobname and the image are replaced by the duplication count followed again by the part number. Thus the difference to short records is the position of the duplication count and the addition of the part number.

In the example above the first and the last record have one line of text followed by blanks. The listing shows part number 1 with the text, followed by the first blank line in part number 2, then 397 duplicates of part number 2 listed in part number 399 and finally the last line with the terminating semicolon.

Please note that if the parameter "`Count duplicates`" had not been set to "`Y`" then each part would have been written and the duplication factors would all be 1.

Log records two and three are listed similarly, except that now there are two text lines and the remaining blanks are in part number 3, then 396 duplicates in part number 399 and then the last one in number 400.

The length of the data records in the application may be variable; the only requirement is that the record be moved to the fixed length parameter that is used in the call to the corresponding logging module.

If an application needs more than 32000-byte data records, then a log module which can handle that can easily be set up. Just increase the length of the parameter from 32000 to the length required up to a maximum of 200000 bytes. If the logging system is called with a larger value then the variable is truncated to 200000 bytes and the return code is set 4, indicating that some of the data was not logged.

Of course it is possible also to generate a logging module where the length of the log-record is variable, supplied by the application in the linkage area. The definition of the variable should be as follows:

```
       01 Log-Record-Length        PIC S9(8) BINARY
```

and the code for the call to SDLTRACE should be changed from:

```
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H      BY CONTENT LENGTH OF Log-Record
```

to:

```
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H                           Log-Record-Length
```

The fixed length of the log record defined in the log module is thus replaced by the variable value supplied by the caller. For an example please look at LOGMOD32:

```
       Identification Division.
       Program-ID. LOGMOD32.

       Data Division.
       Working-Storage Section.
       01 Log-Record           pic x(32000).

       Linkage Section.
       01 Log-Data             pic x(32000).
       01 Log-Data-Length      pic s9(8) binary.

       Procedure Division using Log-Data Log-Data-Length.
       Main.
           move Log-Data to Log-Record
           goback.

       End program LOGMOD32.
```

After processing this program with ATRACE using the same panel values as for LOGMOD31 we get the following logging code:

```
       .
       .
       01 Log-Record           pic x(32000).

       Linkage Section.
       01 Log-Data             pic x(32000).
       01 Log-Data-Length      pic s9(8) binary.

       Procedure Division using Log-Data Log-Data-Length.
       .
       .
       Main.
           move Log-Data to Log-Record
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'Log-Record' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H      BY CONTENT LENGTH OF Log-Length
SDL#E      END-CALL
```

The MOVE statement that transfers the data from the LINKAGE AREA data to the internal variable is actually only used to trigger the generation of the SDLTRACE code and determine the length of the log data. Since the length is supplied in the call we can now modify the generated code by deleting the definition of `Log-Record` and the corresponding MOVE statement and change the CALL to use the variable `Log-Data` instead:

```
       .
       .
       Linkage Section.
       01 Log-Data             pic x(32000).
       01 Log-Data-Length      pic s9(8) binary.

       Procedure Division using Log-Data Log-Data-Length.
       .
       .
       Main.
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'Log-Record' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Data
SDL#H                           Log-Data-Length
SDL#E      END-CALL
       .
       .
```

The complete module can be found in library *"user-id"*.SDLTRACE.DEMO.COBOL under the name LOGMXD32. Please submit it for compilation and linking to produce the load module LOGMOD32.

A sample program to test this log module is SAMPLG32:

```
       ID Division.
       Program-ID  SAMPLG32.

       Data Division.

       Working-Storage Section.
       01  log-data                  pic  x(32000) value space.
       01  log-data-length           pic s9(8)  binary value 80.
       01  log-count                 pic  9(5)  value zero.
       01  log-routine               pic  x(8)  value 'LOGMOD32'.

       Procedure Division.
           move '-- Logging example for large records with variable leng
      -         'th data --'   to  log-data
           call log-routine using  log-data  log-data-length
           move 'Record number 00000. The data to be logged may be of an
      -         'y size, as long as it is an exact multiple of 80. This
      -         'is an example with size 00000000.'  to log-data
           perform varying log-count from 1 by 1 until log-count > 4
             move log-count       to log-data(15:5)
             compute log-data-length = 80 + 80 * log-count
             move log-data-length to log-data(135:8)
             call log-routine  using log-data log-data-length
           end-perform
           move '-- Last record, variable test complete --' to log-data
           move 80 to log-data-length
           call log-routine using log-data log-data-length
           goback.

       End Program SAMPLG32.
```

When this program is run a log dataset with the following contents will be generated:

```
-- Logging example for large records with variable length data --
Record number 00001. The data to be logged may be of any size, as long as it is
an exact multiple of 80. This is an example with size 00000160.
Record number 00002. The data to be logged may be of any size, as long as it is
an exact multiple of 80. This is an example with size 00000240.

Record number 00003. The data to be logged may be of any size, as long as it is
an exact multiple of 80. This is an example with size 00000320.


Record number 00004. The data to be logged may be of any size, as long as it is
an exact multiple of 80. This is an example with size 00000400.



-- Last record, variable test complete --
```

with the rightmost 53 columns:

```
                           SAMPLG32SAMPLG32E003   1 L 2015-07-12 12:43:54.731511
                           SAMPLG32SAMPLG32E003   1,L 2015-07-12 12:43:54.767043
                           SAMPLG32     1         2;L 2015-07-12 12:43:54.767043
                           SAMPLG32SAMPLG32E003   1,L 2015-07-12 12:43:54.767070
                           SAMPLG32     1         2.L 2015-07-12 12:43:54.767070
                           SAMPLG32     1         3;L 2015-07-12 12:43:54.767070
                           SAMPLG32SAMPLG32E003   1,L 2015-07-12 12:43:54.767097
                           SAMPLG32     1         2.L 2015-07-12 12:43:54.767097
                           SAMPLG32     1         3.L 2015-07-12 12:43:54.767097
                           SAMPLG32     1         4;L 2015-07-12 12:43:54.767097
                           SAMPLG32SAMPLG32E003   1,L 2015-07-12 12:43:54.767130
                           SAMPLG32     1         2.L 2015-07-12 12:43:54.767130
                           SAMPLG32     1         3.L 2015-07-12 12:43:54.767130
                           SAMPLG32     1         4.L 2015-07-12 12:43:54.767130
                           SAMPLG32     1         5;L 2015-07-12 12:43:54.767130
                           SAMPLG32SAMPLG32E003   1 L 2015-07-12 12:43:54.767173
```

Again, just as in the previous examples, the generation of the modules described above may be performed in batch mode automatically, without having to enter any data into the ATRACE panel manually. Please display the relevant members by specifying on the ISPF DSN display:

```
e        user-id.SDLTRACE.DEMO.CNTL(SDLPRE3*)
```

The following members will be shown:

```
_________ SDLPRE31
_________ SDLPRE32
```

Just submit `SDLPRE31` and `SDLPRE32` by entering “j” on the command line. This will use the ATRACE panel parameters defined in `SDLPRC31` to generate the logging module `LOGMOD31` as well as `LOGMOD32`.

The sample jobs in the COBOL library which use these modules will be displayed when specifying:

```
e        user-id.SDLTRACE.DEMO.COBOL(SAMPLG3*)
```

on the ISPF DSN display. The following members should be shown:

```
_________ SAMPLG31
_________ SAMPLG32
```

These jobs can now be run to generate the example log datasets.

So far we have created four different logging modules:

LOGMOD1x – log modules that need only one parameter: the 80-byte record to be logged.
LOGMOD2x – log modules which require the application-ID as second parameter.
LOGMOD31 – log module with one parameter and log-data that may be up to 32000 bytes long.
LOGMOD32 – log module with two parameters: log-data that may be up to 32000 bytes long
and the (variable) length of the log-data.

In the next chapter we will set up a log module which will allow us to measure execution times.

Chapter 4. Measuring execution times

Together with each log record the date and time of its creation are kept, either in local or in universal time (GMT), denoted by a blank (the default) or an asterisk "*" in front of the timestamp. These times can be used to compute the difference between individual events and thus arrive at elapsed times for any interval of special interest.

However, with the built-in timing feature it is much easier to get any timing information desired.
By placing appropriate statements at relevant points in a program it is possible to measure the elapsed times between any number of pairs of start and stop locations. The following explanations will show how to set up a program for measuring its execution times.

Just as in the previous examples please go to library "*user-id"*.SDLTRACE.DEMO.COBOL and look at module LOGMOD41:

```
Identification Division.

Program-ID. LOGMOD41.

Data Division.
Working-Storage Section.
01 Log-Record           pic x(80).

Linkage Section.
01 Log-Data             pic x(80).

Procedure Division using Log-data.
Main.
    move Log-Data to Log-Record
    move Log-Record to Log-Data
    goback.

End program LOGMOD41.
```

This program should be processed with the following ATRACE panel to convert it to a real logging module, where the two move statements will finally make sense:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                          Screen 7 of 7
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        LOGMOD41
Output member       LOGMXD41        Insert/Remove/Edit I (I/R/E)

DSN qualifier       user-id         Trace/Log mode     L (T/L)
Application-ID      SDLAPPL4        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)  0___ (0-1440)   Trace variables    Y (Y/N)

Count duplicates    N (Y/N)         Include string #1  LOG-RECORD____________
Console messages    Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE    Y (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       Y (Y/N)         Exclude string #3  ______________________
Timing threshold ____0 (0-32767 ms) Pgm1 ________ Pgm2 ________ Pgm3 ________
```

```
F3 = Quit                                                 ENTER = Process input
```

To do that please go to the library “*user-id”.*SDLTRACE.DEMO.CNTL, step down to member ATRACE and type “**ex**” in front of it. Then please set the values according to the listing shown above. The main difference to all previous panels is the line `Enable timing` , which is set to “`Y`“ . To process the panel, just type “**x**” in the top right entry field and hit “`Enter`”. The following messages will be displayed:

```
SDLTRACE - Version 4.5.23      16 Mar 2015 10:51:46 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - ------------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD41
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD41
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      39
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        126
SDLTRACE - Number of lines with trace:      165
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

Now just hit “`Enter`”; the completed logging module including the timing facility will be displayed.

```
       Identification Division.

       Program-ID. LOGMOD41.

       Data Division.
       Working-Storage Section.
SDL#***------------------------------------------------------------***
SDL#Y  01  SDLTR-PARM  GLOBAL.
SDL#Y      05  SDLTR-VALIDCHK-A   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VERSION      PIC X(8)   VALUE  'VER 4.5 '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-DSN-HILEVEL  PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-APPL-ID      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PGMNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-JOBNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TYPE         PIC X(5)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TEXT         PIC X(50)  VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-START   PIC X(8)   VALUE  'PERFORM '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-END     PIC X(8)   VALUE  '--END-- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-LABEL        PIC X(8)   VALUE  '------- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PRI-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-SEC-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-VAR-LENGTH   PIC 9(4)   VALUE 31   BINARY.
```

```
SDL#Y      05  SDLTR-FLD-LENGTH   PIC 9(4)   VALUE 13   BINARY.
SDL#Y      05  SDLTR-THRESHOLD    PIC 9(4)   VALUE 0    BINARY.
SDL#Y      05  SDLTR-NEWTIM       PIC 9(4)   VALUE 1440 BINARY.
SDL#Y      05  SDLTR-RETN-CODE    PIC 9(2)   VALUE   0.
SDL#Y      05  SDLTR-RETN-CBIN    PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TRACE        PIC 9(1)   VALUE   1.
SDL#Y        88 SDLTR-TRACE-ON               VALUE   1.
SDL#Y        88 SDLTR-TRACE-OFF              VALUE   0.
SDL#Y      05  SDLTR-CALLER       PIC X(1)   VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-ASM             VALUE   'A'.
SDL#Y        88 SDLTR-CALLER-COBOL           VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-C               VALUE   'C'.
SDL#Y      05  SDLTR-VAR-TYPE     PIC X(1)   VALUE   SPACE.
SDL#Y        88 SDLTR-VAR-TYPE-ALL           VALUE   ' '.
SDL#Y        88 SDLTR-VAR-TYPE-BIN           VALUE   'B'.
SDL#Y        88 SDLTR-VAR-TYPE-CHR           VALUE   'C'.
SDL#Y        88 SDLTR-VAR-TYPE-DEC           VALUE   'D'.
SDL#Y        88 SDLTR-VAR-TYPE-HEX           VALUE   'X'.
SDL#Y      05  SDLTR-TIMESTAMP    PIC X(1)   VALUE   'L'.
SDL#Y        88 SDLTR-TMSTP-GMT              VALUE   'G'.
SDL#Y        88 SDLTR-TMSTP-LOC              VALUE   'L'.
SDL#Y      05  SDLTR-WRITE-IMM    PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-WRITE-IMM-ON           VALUE    1.
SDL#Y        88 SDLTR-WRITE-IMM-OFF          VALUE    0.
SDL#Y      05  SDLTR-CONS-MSG-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-CONS-MSG-OFF           VALUE    1.
SDL#Y        88 SDLTR-CONS-MSG-ON            VALUE    0.
SDL#Y      05  SDLTR-TIMING-IND   PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TIMING-ON              VALUE    1.
SDL#Y        88 SDLTR-TIMING-OFF             VALUE    0.
SDL#Y      05  SDLTR-TRACECTL     PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TRACECTL-ON            VALUE    1.
SDL#Y        88 SDLTR-TRACECTL-OFF           VALUE    0.
SDL#Y      05  SDLTR-LOG-MODE-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-LOG-MODE-ON            VALUE    1.
SDL#Y        88 SDLTR-LOG-MODE-OFF           VALUE    0.
SDL#Y      05  SDLTR-DUPLICAT-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-DUPLICAT-ON            VALUE    1.
SDL#Y        88 SDLTR-DUPLICAT-OFF           VALUE    0.
SDL#Y      05  SDLTR-SKIP-NAME1   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME2   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME3   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-RESERVED     PIC X(18)  VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SYSTEM-AREA  PIC X(1800) VALUE  LOW-VALUE.
SDL#Y      05  SDLTR-VALIDCHK-Z   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y
SDL#Y  01  SDLTR-WORK  GLOBAL.
SDL#Y      05  SDLTR-LENGTH       PIC S9(9) BINARY.
SDL#Y      05  SDLTR-SAVERC       PIC S9(4) BINARY.
SDL#Y      05  SDLTR-INDEX1       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX2       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX1-NUM   PIC 9(8).
SDL#Y      05  SDLTR-INDEX2-NUM   PIC 9(8).
SDL#Y      05  SDLTR-SET-TRUE     PIC X(4)  VALUE 'TRUE'.
SDL#Y      05  SDLTR-GEN-DATE     PIC X(11) VALUE '2 Jul 2015'.
SDL#Y      05  SDLTRACE           PIC X(8)  VALUE 'SDLTRACE'.
SDL#***------------------------------------------------------------***
       01 Log-Record           pic x(80).

       Linkage Section.
       01 Log-Data             pic x(80).
       Procedure Division using Log-data.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'LOGMOD41' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE 'SDLAPPL4' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 0          TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-ON   TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-OFF  TO TRUE
SDL#K      SET  SDLTR-TIMING-ON     TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE.
SDL#D      .
SDL#***------------------------------------------------------------***
       Main.
           move Log-Data to Log-Record
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'Log-Record' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H      BY CONTENT LENGTH OF Log-Record
SDL#E      END-CALL
SDL#O      MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
           move Log-Record to Log-Data
SDL#***------------------------------------------------------------***
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#Z *    MOVE 'STOP '    TO SDLTR-TYPE
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
           goback.

       End program LOGMOD41.
```

Please submit the job above to compile and link it into the library *“user-id”.*SDLTRACE.DEMO.LOAD.

To test LOGMOD41 the program SAMPLG41 in library *“user-id”.*SDLTRACE.DEMO.COBOL has been provided:

```
       ID Division.

       Program-ID  SAMPLG41.

       Data Division.

       Working-Storage Section.

       01  log-data.
```

```
  05  log-rec              pic  x(13) value 'Log record # '.
  05  log-count            pic  9(5)  value zero.
  05  log-fill             pic  x(62) value space.
01  log-routine            pic  x(8)  value 'LOGMOD41'.

Procedure Division.

    perform varying log-count from 1 by 1 until log-count > 100
      call log-routine  using  log-data
    end-perform
    goback.

end program SAMPLG41.
```

When this program is being run, a log dataset with 100 records is created. The first 15 lines look like this:

```
Log record # 00001
Log record # 00002                                                     0    38ms
Log record # 00003                                                     0    20µs
Log record # 00004                                                     0    15µs
Log record # 00005                                                     0    15µs
Log record # 00006                                                    81µs  14µs
Log record # 00007                                                     0    15µs
Log record # 00008                                                     0    15µs
Log record # 00009                                                     0    15µs
Log record # 00010                                                     0   171µs
Log record # 00011                                                    82µs  18µs
Log record # 00012                                                     0    22µs
Log record # 00013                                                     0    16µs
Log record # 00014                                                     0    15µs
Log record # 00015                                                     0    16µs
```

with the remaining 53 columns:

```
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.325833
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364437
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364457
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364472
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364487
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364501
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364516
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364531
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364546
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364717
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364735
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364757
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364773
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364788
                           SAMPLG41SAMPLG41E003   1 L 2015-03-16 11:04:58.364804
```

For each log record (except the first one) there are two time values recorded in columns 70 through 80. These times are, however, written into the log record only if there is no logging data present in the space they occupy so that any user data will not be overwritten.

The rightmost time is the elapsed time between two successive calls of SDLTRACE (computed difference between the corresponding timestamps). The time to the left is the used CPU time as recorded by the operating system in the TCB for this particular task. Since the system updates this field only when a task switch takes place, there are many values showing zero, which indicates that the current task was not interrupted by the system (no wait or interrupt, causing a task switch, occurred), nor did another task with higher priority intervene. As a result the elapsed time to the right of a zero CPU value can also be taken as the actual CPU time used.

In the example above the time between successive calls to SDLTRACE is 15 to16 microseconds. As the calling program is not doing anything except to increase the loop counter and then call the logging module again, almost all of the time is being used by the log routine. Therefore the time it takes to record a single log event is about 15 microseconds, which means that for every one million log records to be written the system incurs an additional CPU load of about 15 seconds. Thus there is only a minimal performance impact when using the SDLTRACE logging facility in production.

So far we have just determined the execution times between two successive calls to the log routine.
In order to get the elapsed time between two specific places in the application program, special calls to SDLTRACE have been provided: `'GET TRACE TIMESTAMP'` and `'SET TRACE TIMESTAMP'`.
Please look at the example program SAMPLH41:

```
ID Division.

Program-ID  SAMPLH41.

Data Division.

Working-Storage Section.

01 log-routine            pic  x(8)  value 'LOGMOD41'.
01 pgm-start-time         pic  x(80) value space.

Procedure Division.

    move   'GET TRACE TIMESTAMP'  to  pgm-start-time
    call    log-routine       using  pgm-start-time
    display pgm-start-time
    goback.

end program SAMPLH41.
```

The log routine is called with an 80-byte log record containing the string `'GET TRACE TIMESTAMP'` starting in column 1. This is interpreted by SDLTRACE to **not** write another log record, but rather to establish the start time for the next call where timing information is requested. In addition to setting the start time internally in SDLTRACE the current timestamp is returned to the caller in the log record and can be saved by the application for later use. In order to facilitate that, the `'GET'` is changed to `'SET'`, and the remainder of the record is completed with current time information. This record may be used by the application to set a timing start point anywhere in the program.

When SAMPLH41 is run, no data is written to the logfile; however, a log dataset will nevertheless be allocated to receive data that will be generated in subsequent calls. In this case there are none, so the dataset stays empty. The only action by SDLTRACE is the generation of a timing start point by changing the special log record and returning the data to the caller

The log record for the call to SDLTRACE contains the information:

```
GET TRACE TIMESTAMP
```

The log record returned by SDLTRACE and displayed in SYSOUT will contain the following information:

```
SET TRACE TIMESTAMP 2015-03-16 11:12:15.743899              N!Þ]F€gùµX~F§{é>³ê"&
```

Three changes can be seen;

1. The verb “GET” has been changed to “SET”.
2. In position 21 to 46 the current timestamp has been inserted.
3. Positions 61 to 80 contain this timestamp in binary format.

This log record can be saved by the application program to be used later in a call to the log routine. Similar to `'GET TRACE TIMESTAMP'`, a call to the log routine with `'SET TRACE TIMESTAMP'` is interpreted by SDLTRACE to **not** write another log record. Instead, the start time (that is used to compute the elapsed time) is set to the timestamp (binary format) supplied in the `'SET TRACE TIMESTAMP'` statement. The next call to SDLTRACE will then compute the difference between this timestamp and the current time.

The difference between the two statements `'GET TRACE TIMESTAMP'` and `'SET TRACE TIMESTAMP'` is this: 'GET' sets the timing start point to the current time, whereas 'SET' refers to a previously obtained value.

Please note that only those values that have been obtained through a 'GET' are valid in a 'SET' statement. Therefore the values in a returned 'SET' statement should never be changed.

For an example application program please look at program SAMPLI41:

```
       ID Division.

       Program-ID  SAMPLI41.

       Data Division.

       Working-Storage Section.

       01  start-msg              pic  x(80) value 'Start of program'.
       01  end-msg                pic  x(80) value 'End of program'.
       01  log-data.
         05  log-rec              pic  x(13) value 'Log record # '.
         05  log-count            pic  9(5)  value zero.
         05  log-fill             pic  x(62) value space.
       01  log-routine            pic  x(8)  value 'LOGMOD41'.
       01  wait                   pic  x(8)  value 'SDLWAIT'.
       01  wait-parm.
         05                       pic s9(4)  comp  value 8.
         05  wait-time.
           10  wait-time-hh       pic  9(2)  value  zero.
           10  wait-time-mm       pic  9(2)  value  zero.
           10  wait-time-ss       pic  9(2)  value  zero.
           10  wait-time-th       pic  9(2)  value  10.
       01  pgm-start-time         pic  x(80).
       01  current-time           pic  x(80).

       Procedure Division.

           call  log-routine         using  start-msg
           move 'GET TRACE TIMESTAMP'   to  pgm-start-time
Call 1     call  log-routine         using  pgm-start-time

           perform varying log-count from 1 by 1 until log-count > 10
             move 'GET TRACE TIMESTAMP' to  current-time
Call 2       call  log-routine       using  current-time
             call  wait              using  wait-parm
             call  log-routine       using  log-data
           end-perform

Call 3     call  log-routine   using   pgm-start-time
           call  log-routine   using   end-msg
           goback.

       end program SAMPLI41.
```

In the line marked “Call 1” the start time of the program is obtained and saved for later use.
In “Call 2” the current time is set as start for the computation of the elapsed time for each execution of the statements in the loop. Since the time to wait has been set to 10 hundredths of a second there will be a delay of 100 milliseconds, which is recorded in the log as total elapsed time for each turn through the loop. The time used for the other statements is negligible by comparison. The actual CPU time used is around 38 microseconds, taken from the value stored by the system in the TCB (Task Control Block).

The line marked “Call 3” resets the timestamp to the program start time so that the immediately following call to the log routine can compute the total elapsed time of 1 second. The total CPU time used as stored in the TCB amounts to 484 microseconds.

When this program is run, the following log dataset will be generated:

```
Start of program
Log record # 00001                                                   147µs 105ms
Log record # 00002                                                    40µs 100ms
Log record # 00003                                                    38µs 100ms
Log record # 00004                                                    38µs 100ms
Log record # 00005                                                    38µs 100ms
Log record # 00006                                                    35µs 100ms
Log record # 00007                                                    37µs 100ms
Log record # 00008                                                    37µs 100ms
Log record # 00009                                                    38µs 104ms
Log record # 00010                                                    36µs 100ms
End of program                                                       484µs 1.0s
```

with the remaining columns to the right:

```
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:58.257692
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:58.402735
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:58.503446
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:58.603536
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:58.703599
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:58.803663
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:58.903728
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:59.003937
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:59.104088
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:59.208739
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:59.308816
                          SAMPLI41SAMPLI41E003   1 L 2015-03-16 11:19:59.308840
```

There is a second parameter which can be set when computing execution times: "`Timing threshold`". This parameter specifies a value (in milliseconds) below which a computed elapsed time is not to be recorded in the log record. For example, we want to exclude all values below 200ms from being written to the log file and therefore set up a special logging module, LOGMOD42, accordingly. Please call up ATRACE again and set the panel values as follows:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                             Screen 7 of 7
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         LOGMOD42
Output member        LOGMXD42        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (T/L)
Application-ID       SDLAPPL4        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)   0___ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     N (Y/N)         Include string #1  LOG-RECORD__________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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        Y (Y/N)         Exclude string #3  ______________________
Timing threshold __200 (0-32767 ms)  Pgm1 ________ Pgm2 ________ Pgm3 ________

F3 = Quit                                                ENTER = Process input
```

The values in this panel are all identical to the one used before to create LGOMOD41, except that here the timing threshold is set to 200 milliseconds. After processing this panel and then compiling and linking LOGMOD42 we can run SAMPLG42 which is an exact copy of SAMPLI41, except that it is calling LOGMOD42 instead of LOGMOD41. The resulting log file will show the following:

```
Start of program
Log record # 00001
Log record # 00002
Log record # 00003
Log record # 00004
Log record # 00005
Log record # 00006
Log record # 00007
Log record # 00008
Log record # 00009
Log record # 00010
End of program                                                     449µs 1.0s
```

All elapsed time values below 200ms are being suppressed. Only the last line now contains timing information, in this case 1.0 second for elapsed time and 449 microseconds for CPU time used.

Again, just as in the previous examples, the generation of the modules described above may be performed in batch mode automatically, without having to enter any data into the ATRACE panel manually. Please display the relevant members by specifying on the ISPF DSN display:

```
e        user-id.SDLTRACE.DEMO.CNTL(SDLPRE4*)
```

The following members will be shown:

```
_________ SDLPRE41
_________ SDLPRE42
```

Just submit `SDLPRE41` and `SDLPRE42` by entering “j” on the command line. This will use the ATRACE panel parameters defined in `SDLPRC41` and `SDLPRC42` to generate the logging modules `LOGMOD41` and `LOGMOD42`.

The sample jobs in the COBOL library which use these modules will be displayed when specifying:

```
e        user-id.SDLTRACE.DEMO.COBOL(SAMPL*4*)
```

on the ISPF DSN display. The following members should be shown:

```
_________ SAMPLG41
_________ SAMPLG42
_________ SAMPLH41
_________ SAMPLI41
```

These jobs can now be run to generate the example log datasets described in chapter 4 above.

Chapter 5. Determining the names of calling programs

PLEASE NOTE: The examples in this chapter will not produce the intended results with COBOL 6.1 because of changes to the structure of the save-area made by IBM in the 6.1 release.
See also the Reference Manual, Paragraph 23, regarding the call-tree. (Remark Jan. 14, 2020, HL).

It is often necessary to find out which program calls a specific module or if a certain module is called at all. This can easily be accomplished by inserting a call to logging into the program and specifying the program's name in the ATRACE panel in parameter "`Pgm1`".

By default SDLTRACE inserts the name of the caller of the log routine into positions 81 to 88. Since this is the name of our own program that calls the log routine there is no additional information to be gained. However, the specification of a value in "`Pgm1`" causes SDLTRACE to skip this name and step one level higher up in the call chain and record that name instead. Thus by inserting a call to the log routine into any application program we can determine who called us. A second and a third program name to be skipped may be specified in the fields "`Pgm2`" and "`Pgm3`" .

In order to get the caller of a program we will set up the log module LOGMOD51. Just as in the preceding examples please go to the library *"user-id".*SDLTRACE.DEMO.CNTL, step down to member ATRACE and type "**ex**" in front of it. Then please set the values according to the listing shown below. The main difference to all previous panels is the value for `Pgm1` , which is set to "`SAMPLG51`" .

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                             Screen 7 of 7
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         LOGMOD51
Output member        LOGMXD51        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (T/L)
Application-ID       SDLAPPL5        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)   60__ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     N (Y/N)         Include string #1  LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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 SAMPLG51 Pgm2 ________ Pgm3 ________

F3 = Quit                                               ENTER = Process input
```

Please make sure that the values are indeed as shown above, especially SAMPLG51 in the field `Pgm1` . To process the panel, just type "**x**" in the top right entry field and hit "`Enter`". The following messages will be displayed:

```
SDLTRACE - Version 4.5.23      16 Mar 2015 12:46:09 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - -----------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD51
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD51
SDLTRACE - ------------------------------------
SDLTRACE - Number of lines in original:      38
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        128
SDLTRACE - Number of lines with trace:      166
SDLTRACE - ------------------------------------
SDLTRACE - End of process  RC=0
***
```

Hitting “`Enter`” will display the new version of LOGMOD51, where the only difference to the original log code where no value was specified for `Pgm1` is the following line:

```
SDL#K      MOVE 'SAMPLG51' TO SDLTR-SKIP-NAME1
```

This statement will cause SAMPLG51 to be skipped during the search in the chain of calling programs. Please submit LOGMOD51 now to compile and link it. If the caller of this log module is SAMPLG51 then SDLTRACE will look one step higher in the call chain and record that name instead.

For an example program that makes use of LOGMOD51 please look at program SAMPLG51 in library *“user-id”.SDLTRACE.DEMO.COBOL.*

```
       ID Division.

       Program-ID  SAMPLG51.

       Data Division.

       Working-Storage Section.
       01  log-data                 pic  x(80).
       01  log-routine              pic  x(8)  value 'LOGMOD51'.

       Procedure Division.
           move 'Example to determine caller' to log-data
           call log-routine using log-data
           goback.

       End Program SAMPLG51.
```

When this program is run, it will produce a log file with the following data:

```
Example to determine caller
```

with the remaining columns to the right:

```
                            *       SAMPLG51E003   1 L 2015-03-16 12:40:35.323903
```

The program name SAMPLG51 is not recorded as the caller of the log routine. Instead a simple asterisk signifies that a caller of SAMPLG51 could not be found since the operating system itself started it.

Now please look at program SAMPLX51 in library *“user-id”.*SDLTRACE.DEMO.COBOL*.*

```
       ID Division.

       Program-ID  SAMPLX51.

       Data Division.
       Working-Storage Section.
       01  sample                    pic  x(8)  value 'SAMPLG51'.

       Procedure Division.
           call sample
           goback.

       End Program SAMPLX51.
```

There is a simple call to SAMPLG51, and the execution of this program will produce the following log line in the log dataset:

```
Example to determine caller
```

with the remaining columns to the right:

```
                           SAMPLX51SAMPLX51E003   1 L 2015-03-16 12:43:57.846372
```

In this case the caller of SAMPLG51 is SAMPLX51. Any other caller of SAMPLG51 will be recorded similarly and it is thus possible to find all modules that use SAMPLG51. Since the exclusion of the name SAMPLG51 is permanently compiled into the logging module LOGMOD51 the feature in this case is available only for program SAMPLG51. Any other program using LOGMOD51 will therefore not get its caller recorded. This can easily be changed, however, as shown in the following example.

Instead of building the logging module to permanently exclude a specific program name, we set up a logging module which accepts a further parameter similar to a previous logging module where the application-ID was set dynamically. The Cobol variable names to be used for exclusion are called `SDLTR-SKIP-NAME1`, `SDLTR-SKIP-NAME2` and `SDLTR-SKIP-NAME3`.

As an example just look at program LOGMOD52 in library *“user-id”.*SDLTRACE.DEMO.COBOL*:*

```
       Identification Division.

       Program-ID. LOGMOD52.

       Data Division.
       Working-Storage Section.
       01 Log-Record           pic x(80).

       Linkage Section.
       01 Log-Data             pic x(80).
       01 Log-Pgm1             pic x(8).

       Procedure Division using Log-data Log-Pgm1.
       Main.
           move Log-Pgm1   to SDLTR-SKIP-NAME1
           move Log-Data   to Log-Record
           goback.

       End program LOGMOD52.
```

Here we see that in addition to the data to be logged the user is supplying the name of a program that should be skipped by SDLTRACE when it is encountered in the caller-chain search since it is moved to `SDLTR-SKIP-NAME1`. LOGMOD52 so far is only a skeleton, and we still have to complete it before it can be used to log data. Again please go to the library *“user-id”.*SDLTRACE.DEMO.CNTL, step down to member ATRACE and type “**ex**” in front of it. Then please set the values according to the listing shown below.

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                                 Screen 7 of 7
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         LOGMOD52
Output member        LOGMXD52        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (T/L)
Application-ID       SDLAPPL5        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)   60__ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     N (Y/N)         Include string #1  LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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
```

To process the panel, just type "**x**" in the top right entry field and hit "`Enter`". The following messages will be displayed:

```
SDLTRACE - Version 4.5.23      22 Mar 2015 11:07:48 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - -----------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD52
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD52
SDLTRACE - -----------------------------------
SDLTRACE - Number of lines in original:      41
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      168
SDLTRACE - -----------------------------------
SDLTRACE - End of process  RC=0
***
```

Hitting "`Enter`" now will display the completed version of LOGMOD52:

```
       Identification Division.

       Program-ID. LOGMOD52.

       Data Division.
       Working-Storage Section.
SDL#***-------------------------------------------------------------***
SDL#Y  01  SDLTR-PARM  GLOBAL.
SDL#Y      05  SDLTR-VALIDCHK-A   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VERSION      PIC X(8)   VALUE  'VER 4.5 '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-DSN-HILEVEL  PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-APPL-ID      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PGMNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
```

```
SDL#Y      05  SDLTR-JOBNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TYPE         PIC X(5)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TEXT         PIC X(50)  VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-START   PIC X(8)   VALUE  'PERFORM '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-END     PIC X(8)   VALUE  '--END-- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-LABEL        PIC X(8)   VALUE  '------- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PRI-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-SEC-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-VAR-LENGTH   PIC 9(4)   VALUE 31   BINARY.
SDL#Y      05  SDLTR-FLD-LENGTH   PIC 9(4)   VALUE 13   BINARY.
SDL#Y      05  SDLTR-THRESHOLD    PIC 9(4)   VALUE 0    BINARY.
SDL#Y      05  SDLTR-NEWTIM       PIC 9(4)   VALUE 1440 BINARY.
SDL#Y      05  SDLTR-RETN-CODE    PIC 9(2)   VALUE   0.
SDL#Y      05  SDLTR-RETN-CBIN    PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TRACE        PIC 9(1)   VALUE   1.
SDL#Y        88 SDLTR-TRACE-ON               VALUE   1.
SDL#Y        88 SDLTR-TRACE-OFF              VALUE   0.
SDL#Y      05  SDLTR-CALLER       PIC X(1)   VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-ASM             VALUE   'A'.
SDL#Y        88 SDLTR-CALLER-COBOL           VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-C               VALUE   'C'.
SDL#Y      05  SDLTR-VAR-TYPE     PIC X(1)   VALUE   SPACE.
SDL#Y        88 SDLTR-VAR-TYPE-ALL           VALUE   ' '.
SDL#Y        88 SDLTR-VAR-TYPE-BIN           VALUE   'B'.
SDL#Y        88 SDLTR-VAR-TYPE-CHR           VALUE   'C'.
SDL#Y        88 SDLTR-VAR-TYPE-DEC           VALUE   'D'.
SDL#Y        88 SDLTR-VAR-TYPE-HEX           VALUE   'X'.
SDL#Y      05  SDLTR-TIMESTAMP    PIC X(1)   VALUE   'L'.
SDL#Y        88 SDLTR-TMSTP-GMT              VALUE   'G'.
SDL#Y        88 SDLTR-TMSTP-LOC              VALUE   'L'.
SDL#Y      05  SDLTR-WRITE-IMM    PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-WRITE-IMM-ON           VALUE    1.
SDL#Y        88 SDLTR-WRITE-IMM-OFF          VALUE    0.
SDL#Y      05  SDLTR-CONS-MSG-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-CONS-MSG-OFF           VALUE    1.
SDL#Y        88 SDLTR-CONS-MSG-ON            VALUE    0.
SDL#Y      05  SDLTR-TIMING-IND   PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TIMING-ON              VALUE    1.
SDL#Y        88 SDLTR-TIMING-OFF             VALUE    0.
SDL#Y      05  SDLTR-TRACECTL     PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TRACECTL-ON            VALUE    1.
SDL#Y        88 SDLTR-TRACECTL-OFF           VALUE    0.
SDL#Y      05  SDLTR-LOG-MODE-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-LOG-MODE-ON            VALUE    1.
SDL#Y        88 SDLTR-LOG-MODE-OFF           VALUE    0.
SDL#Y      05  SDLTR-DUPLICAT-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-DUPLICAT-ON            VALUE    1.
SDL#Y        88 SDLTR-DUPLICAT-OFF           VALUE    0.
SDL#Y      05  SDLTR-SKIP-NAME1   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME2   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME3   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-RESERVED     PIC X(18)  VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SYSTEM-AREA  PIC X(1800) VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VALIDCHK-Z   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y
```

```
SDL#Y  01  SDLTR-WORK  GLOBAL.
SDL#Y      05  SDLTR-LENGTH       PIC S9(9) BINARY.
SDL#Y      05  SDLTR-SAVERC       PIC S9(4) BINARY.
SDL#Y      05  SDLTR-INDEX1       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX2       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX1-NUM   PIC 9(8).
SDL#Y      05  SDLTR-INDEX2-NUM   PIC 9(8).
SDL#Y      05  SDLTR-SET-TRUE     PIC X(4)   VALUE 'TRUE'.
SDL#Y      05  SDLTR-GEN-DATE     PIC X(11) VALUE '22 Mar 2015'.
SDL#Y      05  SDLTRACE           PIC X(8)   VALUE 'SDLTRACE'.
SDL#***------------------------------------------------------------***
       01 Log-Record           pic x(80).

       Linkage Section.
       01 Log-Data             pic x(80).
       01 Log-Pgm1             pic x(8).

       Procedure Division using Log-data Log-Pgm1.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'LOGMOD52' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE 'SDLAPPL5' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 60         TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-ON   TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-OFF  TO TRUE
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE.
SDL#D      .
SDL#***------------------------------------------------------------***
       Main.
           move Log-Pgm1   to SDLTR-SKIP-NAME1
           move Log-Data   to Log-Record
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'Log-Record' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         Log-Record
SDL#H      BY CONTENT LENGTH OF Log-Record
SDL#E      END-CALL
SDL#O      MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#Z *    MOVE 'STOP '    TO SDLTR-TYPE
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
           goback.

       End program LOGMOD52.
```

Please submit this for compilation and linking. The new source code will be saved under the name of LOGMXD52 automatically when exiting edit mode by hitting PF3.

The program to test this log module is SAMPLG52:

```
ID Division.

Program-ID  SAMPLG52.

Data Division.

Working-Storage Section.
01  log-data                pic  x(80).
01  log-routine             pic  x(8)  value 'LOGMOD52'.
01  skip-pgm-name           pic  x(8)  value 'SAMPLG52'.

Procedure Division.
    move 'Determine caller of caller' to log-data
    call log-routine using log-data skip-pgm-name
    goback.

End Program SAMPLG52.
```

Please run this program; since the caller of LOGMOD52 is SAMPLG52, which is excluded from the search of calling programs, an asterisk '*' is listed to signify that there is no caller (besides the operating system of course). If SAMPLG52 itself is called by another program, then the name of that program will be listed as caller of the logging routine. To check that please run SAMPLX52:

```
ID Division.

Program-ID  SAMPLX52.

Data Division.

Working-Storage Section.
01  sample                  pic  x(8)  value 'SAMPLG52'.

Procedure Division.
    call sample
    goback.

End Program SAMPLX52.
```

Execution of this program will create a logfile which will show SAMPLX52 as caller of the log routine although it is not calling the log module directly, bur rather via SAMPLG52:

```
Determine caller of caller
```

with the remaining columns to the right:

```
                          SAMPLX52SAMPLX52E003   1 L 2015-03-28 10:11:31.442648
```

With the use of LOGMOD52 any application program can use logging to find out by whom it is called. The only requirement is that the application supplies its own name (similar to the code in SAMPLG52) in order to have it skipped during the call chain search. Up to three names may be specified to be skipped in this way by building corresponding log modules, which makes it possible to determine the originator of a call two levels further up in the chain.

Again, just as in the previous examples, the generation of the modules described above may be performed in batch mode automatically without having to enter any data into the ATRACE panel manually. Please display the relevant members by specifying on the ISPF DSN display:

```
e        user-id.SDLTRACE.DEMO.CNTL(SDLPRE5*)
```

The following members will be shown:

```
_________ SDLPRE51
_________ SDLPRE52
```

Just submit SDLPRE51 and SDLPRE52 by entering “j” on the command line. This will use the ATRACE panel parameters defined in SDLPRC51 and SDLPRC52 to generate the logging modules LOGMOD51 and LOGMOD52.

The sample jobs in the COBOL library which use these modules will be displayed when specifying:

```
e        user-id.SDLTRACE.DEMO.COBOL(SAMPL*5*)
```

on the ISPF DSN display. The following members should be shown:

```
_________ SAMPLG51
_________ SAMPLG52
_________ SAMPLX51
_________ SAMPLX52
```

These jobs can now be run to generate the example log datasets described in chapter 5 above.

Chapter 6. Obtaining the current log-record

Sometimes it might be of interest to an application to know what the log-record that has just been written to the logfile looks like, especially the timestamp that was stored. For this purpose there is a special feature available in SDLTRACE logging which is enabled automatically if the log-record is exactly 213 bytes long, rather than the usual 80 bytes. For an example please look at LOGMOD61 in library *"user-id"*.SDLTRACE.DEMO.COBOL:

```
Identification Division.

Program-ID. LOGMOD61.

Data Division.
Working-Storage Section.
01 Log-Record           pic x(213).

Linkage Section.
01 Log-Data.
  05 Log-Rec-Data       pic x(80).
  05 Log-Rec-stored     pic x(133).

Procedure Division using Log-data.
Main.
    move Log-Data   to Log-Record
    move Log-Record to Log-Data
    goback.

End program LOGMOD61.
```

In order to generate the interface to SDLTRACE for program LOGMOD61 please go to the library *"user-id"*.SDLTRACE.DEMO.CNTL, step down to member ATRACE and type "**ex**" in front of it. Then please set the values according to the listing shown below.

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                          Screen 8 of 8
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         LOGMOD61
Output member        LOGMXD61        Insert/Remove/Edit I (I/R/E)

DSN qualifier        user-id         Trace/Log mode     L (T/L)
Application-ID       SDLAPPL6        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)   60__ (0-1440)   Trace variables    Y (Y/N)

Count duplicates     N (Y/N)         Include string #1  LOG-RECORD__________
Console messages     Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE     Y (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
```

To process the panel, just type “**x**” in the top right entry field and hit “`Enter`”. The following messages will be displayed:

```
SDLTRACE - Version 4.5.23      20 Apr 2015 10:32:59 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - -----------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      LOGMOD61
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     LOGMXD61
SDLTRACE - -----------------------------------
SDLTRACE - Number of lines in original:      41
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      168
SDLTRACE - -----------------------------------
SDLTRACE - End of process  RC=0
***
```

Hitting “`Enter`” now will display the completed version of LOGMOD61. Just submit it to compile and link it, then hit SPF3, which will store the source code under the name LOGMXD61. LOGMOD61 is similar to the previous logging modules, except that SDLTRACE will return the contents of the created log record in the field Log-Rec-stored. To check that, please go to program SAMPLG61 in library *“user-id”*.SDLTRACE.DEMO.COBOL:.

```
       ID Division.

       Program-ID  SAMPLG61.

       Data Division.

       Working-Storage Section.
       01 Log-Data.
         05 Log-Rec-Data              pic x(80).
         05 Log-Rec-stored            pic x(133).
       01  log-routine                pic x(8) value 'LOGMOD61'.

       Procedure Division.
           move 'Example to show returned log-record' to Log-Rec-Data
           call log-routine using log-data
           display 'The created log-record is:'
           display Log-Rec-stored
           goback.

       End Program SAMPLG61.
```

When this job is run a logfile will be allocated and the data together with the log information will be stored just as in the previous examples. In addition the job output (SYSOUT) will contain an exact copy of the record that was written to the logfile.

Again, just as in the previous examples, the generation of the modules described above may be performed in batch mode automatically, without having to enter any data into the ATRACE panel manually. Please display the relevant members by specifying on the ISPF DSN display:

```
e        user-id.SDLTRACE.DEMO.CNTL(SDLPRE6*)
```

The following member will be shown:

```
_________ SDLPRE61
```

Just submit `SDLPRE61` by entering “j” on the command line. This will use the ATRACE panel parameters defined in `SDLPRC61` to generate the logging module `LOGMOD61`.

The sample job in the COBOL library which uses this module will be displayed when specifying:

```
e        user-id.SDLTRACE.DEMO.COBOL(SAMPLG6*)
```

on the ISPF DSN display. The following member should be shown:

```
_________ SAMPLG61
```

This job can now be run to generate the example log dataset and the output in SYSOUT described in chapter 6 above.

Chapter 7. Direct logging of an application program

All the examples above used a special log-module to call SDLTRACE, which in most cases is probably the easiest way since most of the trace code is kept separate from the application itself. However, sometimes the setup may be faster without a special log module. To log directly from within an application program we use the same procedure as that for tracing, except that logging is specified as operating mode and a special variable is set up to receive the log data. For an example please look at program SAMPLG71 in library *"user-id".SDLTRACE.DEMO.COBOL:*.

```
       ID DIVISION.

       PROGRAM-ID  SAMPLG71.

       DATA DIVISION.

       WORKING-STORAGE SECTION.
       01  CHARACTER-DATA               PIC  X(20)  VALUE SPACE.
       01  CHARACTER-DATA-LONG          PIC  X(256) VALUE SPACE.
       01  NUMERIC-DATA-UNSIGNED        PIC  9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-POSITIVE PIC S9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-NEGATIVE PIC S9(8)   VALUE ZERO.
       01  DECIMAL-DATA-UNSIGNED        PIC  9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-POSITIVE PIC S9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-NEGATIVE PIC S9(7)   COMP-3 VALUE ZERO.
       01  BINARY-DATA                  PIC S9(9)   BINARY VALUE ZERO.
       01  LOG-RECORD                   PIC  X(80).

       PROCEDURE DIVISION.
           MOVE 'Hello, COBOL!' TO CHARACTER-DATA
           MOVE 'This is character data that extends over more than one
      -         'line and shows how data is displayed on several lines'
                                TO CHARACTER-DATA-LONG
           MOVE  123            TO NUMERIC-DATA-UNSIGNED
           MOVE  456            TO NUMERIC-DATA-SIGNED-POSITIVE
           MOVE -789            TO NUMERIC-DATA-SIGNED-NEGATIVE
           MOVE  123            TO DECIMAL-DATA-UNSIGNED
           MOVE  456            TO DECIMAL-DATA-SIGNED-POSITIVE
           MOVE -789            TO DECIMAL-DATA-SIGNED-NEGATIVE
           MOVE  123456789      TO BINARY-DATA
           MOVE 'SAMPLG71 was executed' TO LOG-RECORD
           GOBACK.

       END PROGRAM SAMPLG71.
```

This program is identical to the first example for tracing, except for the two additional lines: the first one defining LOG-RECORD and the second one moving text into it. Just as for the log modules above we now generate the interface for SDLTRACE. Please go to the library *"user-id".SDLTRACE.DEMO.CNTL*, step down to member ATRACE and type "**ex**" in front of it. Then please set the values according to the listing shown below.

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                                 Screen 9 of 9
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         SAMPLG71
Output member        SAMPLX71        Insert/Remove/Edit I (I/R/E)
```

```
DSN qualifier        user-id         Trace/Log mode      L (T/L)
Application-ID       SDLAPPL7        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)   60__ (0-1440)   Trace variables     Y (Y/N)

Count duplicates     N (Y/N)         Include string #1   LOG-RECORD____________
Console messages     Y (Y/N)         Include string #2   ______________________
Save RETURN-CODE     Y (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
```

To process the panel, just type “**x**” in the top right entry field and hit “`Enter`”. The following messages will be displayed:

```
SDLTRACE - Version 4.5.23      30 May 2015 08:48:13 user-id
SDLTRACE - Parameter file: user-id.SDLTRACE.DEMO.CNTL(SDLPRCTL)
SDLTRACE - Action=INSERT   Mode=LOGGING
SDLTRACE - CALLS will be inserted for:
SDLTRACE - -> SDLTRACE communication area
SDLTRACE - -> Variables after MOVE ... TO
SDLTRACE - -----------------------------------
SDLTRACE - Input library:     user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Input module:      SAMPLG71
SDLTRACE - Output library:    user-id.SDLTRACE.DEMO.COBOL
SDLTRACE - Output module:     SAMPLX71
SDLTRACE - -----------------------------------
SDLTRACE - Number of lines in original:      59
SDLTRACE - MOVE .... TO variables:            1
SDLTRACE - Number of lines inserted:        127
SDLTRACE - Number of lines with trace:      186
SDLTRACE - -----------------------------------
SDLTRACE - End of process  RC=0
***
```

Hitting “`Enter`” now will display the completed version of SAMPLG71:

```
       ID DIVISION.

       PROGRAM-ID  SAMPLG71.

       DATA DIVISION.

       WORKING-STORAGE SECTION.
SDL#***-------------------------------------------------------------***
SDL#Y  01  SDLTR-PARM  GLOBAL.
SDL#Y      05  SDLTR-VALIDCHK-A   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-VERSION      PIC X(8)   VALUE  'VER 4.5 '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-DSN-HILEVEL  PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-APPL-ID      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PGMNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-JOBNAME      PIC X(8)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
```

```
SDL#Y      05  SDLTR-TYPE         PIC X(5)   VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TEXT         PIC X(50)  VALUE   SPACE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-START   PIC X(8)   VALUE  'PERFORM '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PERF-END     PIC X(8)   VALUE  '--END-- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-LABEL        PIC X(8)   VALUE  '------- '.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-PRI-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-SEC-TRKS     PIC 9(4)   VALUE 250  BINARY.
SDL#Y      05  SDLTR-VAR-LENGTH   PIC 9(4)   VALUE 31   BINARY.
SDL#Y      05  SDLTR-FLD-LENGTH   PIC 9(4)   VALUE 13   BINARY.
SDL#Y      05  SDLTR-THRESHOLD    PIC 9(4)   VALUE 0    BINARY.
SDL#Y      05  SDLTR-NEWTIM       PIC 9(4)   VALUE 1440 BINARY.
SDL#Y      05  SDLTR-RETN-CODE    PIC 9(2)   VALUE   0.
SDL#Y      05  SDLTR-RETN-CBIN    PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-TRACE        PIC 9(1)   VALUE   1.
SDL#Y        88 SDLTR-TRACE-ON               VALUE   1.
SDL#Y        88 SDLTR-TRACE-OFF              VALUE   0.
SDL#Y      05  SDLTR-CALLER       PIC X(1)   VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-ASM             VALUE   'A'.
SDL#Y        88 SDLTR-CALLER-COBOL           VALUE   'B'.
SDL#Y        88 SDLTR-CALLER-C               VALUE   'C'.
SDL#Y      05  SDLTR-VAR-TYPE     PIC X(1)   VALUE   SPACE.
SDL#Y        88 SDLTR-VAR-TYPE-ALL           VALUE   ' '.
SDL#Y        88 SDLTR-VAR-TYPE-BIN           VALUE   'B'.
SDL#Y        88 SDLTR-VAR-TYPE-CHR           VALUE   'C'.
SDL#Y        88 SDLTR-VAR-TYPE-DEC           VALUE   'D'.
SDL#Y        88 SDLTR-VAR-TYPE-HEX           VALUE   'X'.
SDL#Y      05  SDLTR-TIMESTAMP    PIC X(1)   VALUE   'L'.
SDL#Y        88 SDLTR-TMSTP-GMT              VALUE   'G'.
SDL#Y        88 SDLTR-TMSTP-LOC              VALUE   'L'.
SDL#Y      05  SDLTR-WRITE-IMM    PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-WRITE-IMM-ON           VALUE    1.
SDL#Y        88 SDLTR-WRITE-IMM-OFF          VALUE    0.
SDL#Y      05  SDLTR-CONS-MSG-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-CONS-MSG-OFF           VALUE    1.
SDL#Y        88 SDLTR-CONS-MSG-ON            VALUE    0.
SDL#Y      05  SDLTR-TIMING-IND   PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TIMING-ON              VALUE    1.
SDL#Y        88 SDLTR-TIMING-OFF             VALUE    0.
SDL#Y      05  SDLTR-TRACECTL     PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-TRACECTL-ON            VALUE    1.
SDL#Y        88 SDLTR-TRACECTL-OFF           VALUE    0.
SDL#Y      05  SDLTR-LOG-MODE-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-LOG-MODE-ON            VALUE    1.
SDL#Y        88 SDLTR-LOG-MODE-OFF           VALUE    0.
SDL#Y      05  SDLTR-DUPLICAT-IND PIC 9(1)   VALUE    0.
SDL#Y        88 SDLTR-DUPLICAT-ON            VALUE    1.
SDL#Y        88 SDLTR-DUPLICAT-OFF           VALUE    0.
SDL#Y      05  SDLTR-SKIP-NAME1   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME2   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SKIP-NAME3   PIC X(8)   VALUE   LOW-VALUE.
SDL#Y      05                     PIC X(1)   VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-RESERVED     PIC X(18)  VALUE   LOW-VALUE.
SDL#Y      05  SDLTR-SYSTEM-AREA  PIC X(1800) VALUE  LOW-VALUE.
SDL#Y      05  SDLTR-VALIDCHK-Z   PIC X(8)   VALUE  'SDLTRACE'.
SDL#Y
SDL#Y  01  SDLTR-WORK  GLOBAL.
SDL#Y      05  SDLTR-LENGTH       PIC S9(9) BINARY.
```

```
SDL#Y      05  SDLTR-SAVERC       PIC S9(4) BINARY.
SDL#Y      05  SDLTR-INDEX1       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX2       PIC ZZZZZZZ9.
SDL#Y      05  SDLTR-INDEX1-NUM   PIC 9(8).
SDL#Y      05  SDLTR-INDEX2-NUM   PIC 9(8).
SDL#Y      05  SDLTR-SET-TRUE     PIC X(4)  VALUE 'TRUE'.
SDL#Y      05  SDLTR-GEN-DATE     PIC X(11) VALUE '30 May 2015'.
SDL#Y      05  SDLTRACE           PIC X(8)  VALUE 'SDLTRACE'.
SDL#***------------------------------------------------------------***
       01  CHARACTER-DATA               PIC  X(20)  VALUE SPACE.
       01  CHARACTER-DATA-LONG          PIC  X(256) VALUE SPACE.
       01  NUMERIC-DATA-UNSIGNED        PIC  9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-POSITIVE PIC S9(8)   VALUE ZERO.
       01  NUMERIC-DATA-SIGNED-NEGATIVE PIC S9(8)   VALUE ZERO.
       01  DECIMAL-DATA-UNSIGNED        PIC  9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-POSITIVE PIC S9(7)   COMP-3 VALUE ZERO.
       01  DECIMAL-DATA-SIGNED-NEGATIVE PIC S9(7)   COMP-3 VALUE ZERO.
       01  BINARY-DATA                  PIC S9(9)   BINARY VALUE ZERO.
       01  LOG-RECORD                   PIC  X(80).

       PROCEDURE DIVISION.
SDL#***------------------------------------------------------------***
SDL#A  START-TRACE-INITIALIZATION.
SDL#S      MOVE 'START'    TO SDLTR-TYPE
SDL#N      MOVE 'SAMPLG71' TO SDLTR-PGMNAME
SDL#K      MOVE 'user-id ' TO SDLTR-DSN-HILEVEL
SDL#K      MOVE 'SDLAPPL7' TO SDLTR-APPL-ID
SDL#K      MOVE '*       ' TO SDLTR-JOBNAME
SDL#K      MOVE 0          TO SDLTR-THRESHOLD
SDL#K      MOVE 100        TO SDLTR-PRI-TRKS
SDL#K      MOVE 100        TO SDLTR-SEC-TRKS
SDL#K      MOVE 60         TO SDLTR-NEWTIM
SDL#K      SET  SDLTR-WRITE-IMM-OFF TO TRUE
SDL#K      SET  SDLTR-TRACECTL-OFF  TO TRUE
SDL#K      SET  SDLTR-CONS-MSG-ON   TO TRUE
SDL#K      SET  SDLTR-LOG-MODE-ON   TO TRUE
SDL#K      SET  SDLTR-DUPLICAT-OFF  TO TRUE
SDL#K      SET  SDLTR-TIMING-OFF    TO TRUE
SDL#K      SET  SDLTR-TMSTP-LOC     TO TRUE
SDL#K      SET  SDLTR-TRACE-ON      TO TRUE
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE.
SDL#D      .
SDL#***------------------------------------------------------------***
           MOVE 'Hello, COBOL!' TO CHARACTER-DATA
           MOVE 'This is character data that extends over more than one
      -         'line and shows how data is displayed on several lines'
                                TO CHARACTER-DATA-LONG
           MOVE  123            TO NUMERIC-DATA-UNSIGNED
           MOVE  456            TO NUMERIC-DATA-SIGNED-POSITIVE
           MOVE -789            TO NUMERIC-DATA-SIGNED-NEGATIVE
           MOVE  123            TO DECIMAL-DATA-UNSIGNED
           MOVE  456            TO DECIMAL-DATA-SIGNED-POSITIVE
           MOVE -789            TO DECIMAL-DATA-SIGNED-NEGATIVE
           MOVE  123456789      TO BINARY-DATA
           MOVE 'SAMPLG71 was executed' TO LOG-RECORD
SDL#***------------------------------------------------------------***
SDL#R      MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#I      MOVE 'V1'       TO SDLTR-TYPE
SDL#F      MOVE 'LOG-RECORD' TO SDLTR-TEXT
SDL#C      CALL  SDLTRACE  USING SDLTR-PARM
SDL#V      BY REFERENCE         LOG-RECORD
SDL#H      BY CONTENT LENGTH OF LOG-RECORD
```

```
SDL#E      END-CALL
SDL#O      MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
SDL#***------------------------------------------------------------***
SDL#R *    MOVE RETURN-CODE TO SDLTR-SAVERC
SDL#Z *    MOVE 'STOP '    TO SDLTR-TYPE
SDL#C *    CALL SDLTRACE USING SDLTR-PARM END-CALL
SDL#O *    MOVE SDLTR-SAVERC TO RETURN-CODE
SDL#***------------------------------------------------------------***
           GOBACK.

       END PROGRAM SAMPLG71.
```

Just submit the program above for compilation, linking and execution, then hit SPF3 to store the source code under the name SAMPLGX71 into the library. *"user-id".*SDLTRACE.DEMO.COBOL*.*
Since the GO-step was included in the JCL the program will be executed right away and produces a log-dataset with the following contents:

```
SAMPLG71 was executed
                           SAMPLG71SAMPLG71E003   1 L 2015-05-30 09:10:14.431773
```

It is also possible to combine tracing and logging within one program. Take for example the program SAMPLG72. It is identical to SAMPLG71, except that we now apply the values from the following ATRACE panel:

```
SDLTRACE V4.5  09/13/05         COBOL PREPROCESSOR
                                                            Screen 9 of 9
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        SAMPLG72
Output member       SAMPLX72        Insert/Remove/Edit I (I/R/E)

DSN qualifier       user-id         Trace/Log mode     L (T/L)
Application-ID      SDLAPPL7        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)  60__ (0-1440)   Trace variables    Y (Y/N)

Count duplicates    N (Y/N)         Include string #1  ______________________
Console messages    Y (Y/N)         Include string #2  ______________________
Save RETURN-CODE    Y (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
```

The only difference to the previous panel for SAMPLG71 is the specification line:

```
Include string #1  ______________________
```

which now does not contain anything, whereas before we specified the variable `LOG-RECORD` in order to limit the inclusion of only those move statements that involve `LOG-RECORD` . As a result all variable changes will trigger the inclusion of code to call SDLTRACE, and although log-mode is specified, only those variables which are exactly 80 bytes long will be treated as log entries; all others are handled as if we were in trace mode. Thus execution of SAMPLG72 after processing by ATRACE will show the following contents in the log dataset:

```
V1    CHARACTER-DATA                  Hello, COBOL!
```

```
V2    CHARACTER-DATA-LONG          0   This is character data that extends over
V2                                41 *  more than one line and shows how data i
V2                                82 * s displayed on several lines
V2                               123 *
V2    =       3 IDENTICAL LINES - - - - - - - - - - - - - - - - - - - - - - - - -
V3    NUMERIC-DATA-UNSIGNED             00000123
V4    NUMERIC-DATA-SIGNED-POSITIVE     +00000456
V5    NUMERIC-DATA-SIGNED-NEGATIVE     -00000789
V6    DECIMAL-DATA-UNSIGNED           P 0000123
V7    DECIMAL-DATA-SIGNED-POSITIVE    P+0000456
V8    DECIMAL-DATA-SIGNED-NEGATIVE    P-0000789
V9    BINARY-DATA                     B    123456789
SAMPLG72 was executed
```

with the rightmost part:

```
                           SAMPLG72     1         1 T 2015-05-30 12:43:21.086684
                           SAMPLG72     1         2 T 2015-05-30 12:43:21.132253
                           SAMPLG72     1         3 T 2015-05-30 12:43:21.132253
                           SAMPLG72     1         4 T 2015-05-30 12:43:21.132253
                           SAMPLG72     1         5 T 2015-05-30 12:43:21.132253
 - - - - - - - - - - - - - SAMPLG72     3         8 T 2015-05-30 12:43:21.132253
                           SAMPLG72     1         9 T 2015-05-30 12:43:21.132270
                           SAMPLG72     1        10 T 2015-05-30 12:43:21.132278
                           SAMPLG72     1        11 T 2015-05-30 12:43:21.132286
                           SAMPLG72     1        12 T 2015-05-30 12:43:21.132293
                           SAMPLG72     1        13 T 2015-05-30 12:43:21.132301
                           SAMPLG72     1        14 T 2015-05-30 12:43:21.132308
                           SAMPLG72     1        15 T 2015-05-30 12:43:21.132315
                           SAMPLG72SAMPLG72E003   1 L 2015-05-30 12:43:21.132323
```

Except for the last line which shows the log data, all other lines are standard trace entries just as in the very first example generated by program SAMPLB01. Thus log and trace data may be mixed together, although it is probably better to have separate datasets, one for trace and one for log data. This can be achieved easily by creating a special log module for all logging requests, just as in the examples shown.

Again, just as in the previous examples, the generation of the modules described above may be performed in batch mode automatically, without having to enter any data into the ATRACE panel manually. Please display the relevant members by specifying on the ISPF DSN display:

```
e        user-id.SDLTRACE.DEMO.CNTL(SDLPRE7*)
```

The following members will be shown:

```
_________ SDLPRE71
_________ SDLPRE72
```

Just submit `SDLPRE71` and `SDLPRE72` by entering “j” on the command line. This will use the ATRACE panel parameters defined in `SDLPRC71` and `SDLPRC72` to generate and run the application programs `SAMPLG71` and `SAMPLG72`. Since logging is specified within the user’s program directly, no separate run of an application calling these programs is necessary. The example log datasets described in chapter 7 above will have been generated as soon as the two jobs are finished.

All log modules required for the axamples described above can also be generated by submitting the job `SDLPRE00` out of library `user-id.SDLTRACE.DEMO.CNTL` , and all example jobs may be run by submitting `SDLSARUN`.

Chapter 8. Differences between tracing and logging

Most of the features of SDLTRACE apply to both, tracing and logging. The differences between the two modes are in these five areas: Dataset naming conventions, the formatting of variables, the display of execution times, determination of the calling program and the disposition of the trace/log datasets after termination of the current trace/log action.

1. Dataset naming conventions:

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”.

2. Formatting of variables

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).

3. Display of execution times

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).

4. Determination of the calling program

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.

5. Disposition of the trace/log datasets

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 space and time parameters set on the ATRACE panel.