Legac-E Education

Using a Harness to control execution

On LinkedIn in July 2018 a question was raised as to the possibility of controlling the execution of a Job Step from within a COBOL program. The case presented was of a three step job where the second step should be omitted but that for whatever reason an alternative to the use of Condition Codes was being sought.

Obviously language design, Assembler, COBOL, JCL, PLI and REXX to name a few, rely upon the passing of Return Codes which externally become Condition Codes within JCL. What is more there is no easy access to such information within the executing job step. System Management Facility (SMF) will hold information once a step has completed, as will the IEF142I message issue a step end.

Job Design

This question is linked to Job Design as if jobs were created as single step entities reliant upon the Scheduler for the correct processing sequence there would be few if any issues. This aspect was discussed in:

www.legac-e.co.uk/JCLdocs/jobdes.pdf

If forced to make a recommendation then the single step per job approach would be it as it avoids the issue raised and makes re-run situations much less complicated.

The Internal Reader (INTRDR)

Technically there is the potential to use the Internal Reader (INTRDR) to have the first program of a sequence submit subsequent programs as separate jobs, but this lacks a holistic approach. Operations are responsible for managing the system and ensuring that the correct jobs are run in the relevant sequence at the correct time and they rely on a Scheduling Package such as Tivoli Workload Scheduler (TWS), CA-7, or CONTROL-M, to aid them. Computer Auditors typically raise issue if production work is submitted outside of Scheduler control. **Encouraging the submission of jobs outside of the Scheduler is akin to sanctioning development outside of Endeavor or equivalent and should not be recommended**.

Using a Harness to control execution

A Harness

The concept of a harness is that the multi-step job is converted to a single step job with all the DD statements of the programs which constituted the multiple steps. The single EXEC statement names the harness program which calls the other programs based on criteria specified by the use. This does not contravene the principle of submitting work via the Scheduler as it would still track the overall job as usual. The initial challenges with this approach are likely to be:

- DD Statement conflicts such as the same name is used by different programs for different files
- Multiple use of the same DDNAME for the same purpose but which might cause data loss, i.e. repeated opening of SYSPRINT
- How to restart at a particular point within the sequence

An early incarnation of such a program generated a false sense of ease, but there are significant architecture difference now compared to the early 1970s when everything ran in 24-bit mode so additional challenges became evident:

- Parameter lists for IBM Utilities need to be in 24-bit storage
- The default addressing mode (AMODE) for COBOL with Language Environment (LE) is 31-bit both AMODE and RMODE.
- The use of VSAM files particularly if DELETE and DEFINE are contemplated dynamically

There may be other challenges which were not identified in the experimentation that was undertaken and described in the rest of this document.

What is clear is that producing a harness program is feasible but is not something that would be recommended.

Using a Harness to control execution

Overcoming the challenges

DDNAME SYSOUT

This DDNAME is used by both COBOL and LE for diagnostic purposes and the recording of messages produced by COBOL DISPLAY. There is no conflict here once the data set has been opened subsequent messages are written in sequence irrespective of source so there is no conflict. The following is therefore valid for all requirements:

```
//SYSOUT DD SYSOUT=*
```

DDNAME SYSPRINT

The problem here is that any called program which issues an OPEN / CLOSE sequence for SYSPRINT may overwrite any output produced by a preceding program which performs the same operation. To overcome this, two actions were taken

1. Dynamically invoke IEBGENER with alternate DDNAMES as part of termination so that accumulated SYSPRINT could be written out to a new report file
2. Include a SYSPRINT DD Statement like:

```
//SYSPRINT DD DISP=(MOD,PASS),SPACE=(TRK,5),
//         LRECL=125,RECFM=VBA,BLKSIZE=1254
```

DDNAME SYSIN

The test requirement involved IDCAMS being called twice and IEBGENER once therefore the conflict arising from SYSIN was dealt with thus;

1. The IEBGENER DDNAME list was updated to use GENSYSIN rather than SYSIN
2. Each IDCAMS SYSIN was given an alternative name, IDCAMS01 for the first occurrence and IDCAMS03 for the second (logically step 3).
3. The harness program would read the appropriate IDCAMSnn file when required and output the records to a DD Statement like:

```
//SYSIN DD LRECL=80,RECFM=FB,BLKSIZE=0,
//         SPACE=(TRK,5)
```

Using a Harness to control execution

Dynamic VSAM DELETE / DEFINE

With IDCAMS being called twice to DELETE any existing cluster of the desired name, and following it with a DEFINE to create the new cluster there was the potential to have a “DATA SET NOT FOUND” error in respect of the DD statement which would subsequently use the cluster. The first attempt at resolving this used a first step to invoke IEFBR14 to create the necessary data set shells which IDCAMS could DELETE and which would ensure that there was no JCL error. This caused a different problem as by the time the program which needed the cluster ran it was in a different place to the one established at allocation time and hence the programs OPEN failed.

The ultimate resolution was to use Dynamic File allocation for the VSAM file in the called program which meant that both issues were fixed and the IEFBR14 step was not required. The called program was further modified to de-allocate the cluster after CLOSE so that step end was mimicked and the data set was free for other users.

IBM IEB Utility AMODE(24) Parameters

Rather than have a separate IEBGENER step to process the accumulated SYSPRINT data set, IEBGENER was called dynamically but this produced an addressing issue as the old IEB Utility parameters are required to be below the 16MB line, whereas the default AMODE and RMODE for COBOL is 31, i.e. above the 16MB line.

Two actions were taken to ensure there was no conflict:

1. The program was compiled with:

   ```
   CBL DATA(24),RMODE(24)
   ```

2. The program was executed with the LE Run Time options so the PARM looked like:

   ```
    // PARM=’ALL ,S01/ALL31(OFF),STACK(,,BELOW)
   ```

Using a Harness to control execution

The Sample Harness Program

This is not a fully functioning harness as it only provides for two scenarios, either running all the programs in the schedule or simply running a single program. The capability to omit a program is not present other than to simply delete it from the SCHEDULE file or replace its name with IEFBR14.

```
 CBL DATA(24),RMODE(24)
 IDENTIFICATION DIVISION.
 PROGRAM-ID.                     HARNESS
 AUTHOR.                         T.R.SAMBROOKS.
     INSTALLATION.
     DATE-WRITTEN.               22nd jul 2018.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*--------------------------------------------------------------*
* Program to demonstrate effective omission of Job Steps.     *
*--------------------------------------------------------------*
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT PGM-LIST             ASSIGN TO UT-S-SCHEDULE.
     SELECT DYN-FILE             ASSIGN TO UT-S-DYNFILE.
     SELECT CNTL-FILE            ASSIGN TO UT-S-SYSIN.
     SELECT SYSPRINT-IN          ASSIGN TO UT-S-SYSPRINT.
     SELECT STEPMSGS-OUT         ASSIGN TO UT-S-STEPMSGS.
 DATA DIVISION.
 FILE SECTION.
 FD  PGM-LIST                    RECORDING MODE IS F
                                 LABEL RECORDS ARE STANDARD
                                 BLOCK CONTAINS 0 RECORDS
                                 RECORD CONTAINS 80 CHARACTERS
                                 DATA RECORD IS PGM_REC.
 01  PGM-REC.
     03  STEP-NAME               PIC X(3).
     03                          PIC X.
     03  PGM-NAME                PIC X(8).
     03                          PIC X.
     03  GOOD-CC                 PIC 99.
     03                          PIC X.
     03  PARM-DD                 PIC X(8).
     03                          PIC X.
     03  PARM-DSN                PIC X(54).
     03                          PIC X.
```

Using a Harness to control execution

```
FD  DYN-FILE                    RECORDING MODE IS F
                                LABEL RECORDS ARE STANDARD
                                BLOCK CONTAINS 0 RECORDS
                                RECORD CONTAINS 80 CHARACTERS
                                DATA RECORD IS DYN_REC.
01  DYN-REC                     PIC X(80).
FD  CNTL-FILE                   RECORDING MODE IS F
                                LABEL RECORDS ARE STANDARD
                                BLOCK CONTAINS 0 RECORDS
                                RECORD CONTAINS 80 CHARACTERS
                                DATA RECORD IS DYN_REC.
01  CNTL-REC                    PIC X(80).
FD  SYSPRINT-IN                 RECORDING MODE IS V
                                LABEL RECORDS ARE STANDARD
                                BLOCK CONTAINS 0 RECORDS
                                RECORD VARYING 50 TO 121
                                       DEPENDING ON PRINT-RDW
                                DATA RECORD IS SYSPRINT-REC.
01  SYSPRINT-REC.
    03  PRINT-LINE.
        05                      PIC X(50).
        05                      PIC X OCCURS 1 TO 70
                                      DEPENDING ON PRINT-RDW.
FD  STEPMSGS-OUT                RECORDING MODE IS F
                                LABEL RECORDS ARE STANDARD
                                BLOCK CONTAINS 0 RECORDS
                                RECORD CONTAINS 121
                                DATA RECORD IS STEPMSGS-REC.
01  STEPMSG-REC                 PIC X(121).
WORKING-STORAGE SECTION.
01  WS-ADHOC-CONSTANTS.
    03  PRINT-RDW               PIC 9(8) COMP.
    03  SAVED-CC                PIC S9(4) COMP VALUE 0.
    03  STEP-COUNT              PIC S9(4) COMP VALUE 0.
    03  ISUB                    PIC S9(4) COMP VALUE +1.
    03  RSUB                    PIC S9(4) COMP VALUE +1.
    03  UTIL-PGM                PIC X(8) VALUE 'IEBGENER'.
    03  DYN-PGM                 PIC X(8) VALUE 'BPXWDYN '.
    03  EOF-DYNFILE             PIC X          VALUE 'R'.
      88  DYN-DONE              VALUE 'D'.
    03  EOF-SYSPRINT            PIC X          VALUE 'R'.
      88  ALL-DONE              VALUE 'D'.
    03  EOF-SCHEDULE            PIC X          VALUE 'R'.
      88  SCHED-BUILT           VALUE 'D'.
    03  util-rec                pic x(80).
```

Using a Harness to control execution

```
01  BAD-COND-MSG.
    03                          PIC X(13) VALUE
      '** HARNESS - '.
    03  ERR-PGM                 PIC X(8) VALUE SPACES.
    03                          PIC X(9) VALUE
      ' in step '.
    03  ERR-STEP                PIC X(3) VALUE SPACES.
    03                          PIC X(18) VALUE
      ' issued COND CODE '.
    03  ERR-CC                  PIC 9999.
    03                          PIC X(65) VALUE
      ' other steps bypassed **'.
01  UTIL-PARMS.
    03  OPTLIST                 PIC 9(4) COMP VALUE 0.
    03  DDN-LIST                PIC 9(4) COMP VALUE 72.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03  SYSIN-DDN               PIC X(8) VALUE 'GENSYSIN'.
    03  SYSPRINT-DDN            PIC X(8) VALUE 'GENPRINT'.
    03                          PIC 9(8) COMP VALUE  0.
    03                          PIC 9(8) COMP VALUE  0.
    03  SYSUT1-DDN              PIC X(8) VALUE 'SYSPRINT'.
    03  SYSUT2-DDN              PIC X(8) VALUE 'STEPMSGS'.
    03                          PIC X(8) VALUE 'SYSUT3  '.
    03                          PIC X(8) VALUE 'SYSUT4  '.
    03  HDNGLIST                PIC 9(4) COMP VALUE 2.
    03  PAGENUM                 PIC 9(4) COMP VALUE 1.
01  ALLOC-DYNFILE.
    03                          PIC S9(4) COMP VALUE +88.
    03                          PIC X(9) VALUE
       'ALLOC FI('.
    03  DYN-DDNA                PIC X(7) VALUE 'DYNFILE'.
    03                          PIC X(18) VALUE
       ') SHR MSG(WTP) DA('.
    03  DYN-DSN                 PIC X(54) VALUE SPACES.
01  UNALLOC-DYNFILE.
    03                          PIC S9(4)  COMP VALUE  +26.
    03                          PIC X(8) VALUE
       'FREE FI('.
    03  DYN-DDNU                PIC X(7) VALUE 'DYNFILE'.
    03                          PIC X(2) VALUE ') '.
    03                          PIC X(9) VALUE 'MSG(WTP) '.
```

Using a Harness to control execution

```
 01  DUMMY-SCHEDULE.
     03                          PIC X(76) OCCURS 10 TIMES.
 01  REAL-SCHEDULE               REDEFINES DUMMY-SCHEDULE.
     03  SCHED-ENT               OCCURS 10 TIMES.
         05  NEXT-CC             PIC S9(4) COMP.
         05  NEXT-STEP           PIC X(3).
         05  NEXT-PGM            PIC X(8).
         05  NEXT-PARMDD         PIC X(8).
         05  NEXT-DSN            PIC X(54).
         05                      PIC X.
 LINKAGE SECTION.
 01  EXEC-PARM.
     03  PARM-LEN                PIC S9(4) COMP.
     03  RUN-TYPE                PIC X(4).
         88  ALL-STEPS           VALUE 'ALL '.
         88  ONE-STEP            VALUE 'ONLY'.
         88  RESTART             VALUE 'GOTO'.
     03                          PIC X.
     03  FIRST-STEP              PIC X(3).
 PROCEDURE DIVISION              USING EXEC-PARM.
 A010-HARNESS-SHELL.
     PERFORM B010-INITIALIZE.
     IF NOT ALL-STEPS            PERFORM B030-SET-START.
     PERFORM B020-RUN-SCHEDULE.
     PERFORM B040-TERMINATION.
*--------------------------------------------------------------*
*    Logical end of program - HARNESS.                         *
*--------------------------------------------------------------*
 A010-HARNESS-SHELL-EXIT.
     GOBACK.
 B010-INITIALIZE.
*--------------------------------------------------------------*
*    Build the schedule of programs to be run.                 *
*--------------------------------------------------------------*
     MOVE SPACES                 TO DUMMY-SCHEDULE.
     OPEN INPUT                  PGM-LIST.
     PERFORM                     UNTIL SCHED-BUILT
       READ PGM-LIST            AT END MOVE 'D' TO EOF-SCHEDULE
                                 NOT AT END
                              MOVE GOOD-CC   TO NEXT-CC(ISUB)
                              MOVE STEP-NAME TO NEXT-STEP(ISUB)
                              MOVE PGM-NAME  TO NEXT-PGM(ISUB)
                                 MOVE PARM-DD   TO NEXT-PARMDD
                                                   (ISUB)
                              MOVE PARM-DSN  TO NEXT-DSN (ISUB)
                                 ADD +1         TO ISUB
       END-READ
     END-PERFORM.
```

Using a Harness to control execution

```
     CLOSE PGM-LIST.
     SUBTRACT +1                 FROM ISUB GIVING STEP-COUNT.
     MOVE +1                     TO ISUB.
 B010-INITIALIZE-EXIT.
     EXIT.
 B020-RUN-SCHEDULE.
*-------------------------------------------------------------*
*    Execute the schyedule as requested.                      *
*-------------------------------------------------------------*
     PERFORM                     UNTIL RSUB > STEP-COUNT
       IF NEXT-PARMDD (RSUB)     NOT EQUAL SPACES
                                 PERFORM C020-DYNALLOC
                                 CALL NEXT-PGM (RSUB)
                                 MOVE RETURN-CODE TO SAVED-CC
                                 CALL DYN-PGM USING
                                      UNALLOC-DYNFILE
       ELSE                      CALL NEXT-PGM (RSUB)
                                 MOVE RETURN-CODE TO SAVED-CC
       END-IF
       DISPLAY 'Step ' NEXT-STEP (ISUB) ' Program '
                                    NEXT-PGM (RSUB)
                                    'ended COND CODE '
                                    SAVED-CC
                                    UPON SYSOUT
       IF SAVED-CC > NEXT-CC(ISUB)
                                 PERFORM C010-ERROR-RTN
       END-IF
       ADD +1                    TO RSUB
     END-PERFORM.
 B020-RUN-SCHEDULE-EXIT.
     EXIT.
 B030-SET-START.
*-------------------------------------------------------------*
*    Establish any restart criteria.                          *
*-------------------------------------------------------------*
     PERFORM                     UNTIL ISUB > STEP-COUNT
       IF NEXT-STEP (ISUB) = FIRST-STEP
          EVALUATE RUN-TYPE
            WHEN  'ONLY'         MOVE ISUB TO RSUB
                                 MOVE RSUB TO STEP-COUNT
                                 COMPUTE ISUB =
                                     STEP-COUNT + 1
            WHEN  'GOTO'         MOVE ISUB TO RSUB
                                 ADD +1 TO STEP-COUNT
                                     GIVING ISUB
          END-EVALUATE
       END-IF
     ADD +1                      TO ISUB
```

Using a Harness to control execution

```
          END-PERFORM.
      B030-SET-START-EXIT.
          EXIT.
      B040-TERMINATION.
     *--------------------------------------------------------------*
     *    Print accumulated SYSPRINT messages from any utilities.   *
     *--------------------------------------------------------------*
          CALL UTIL-PGM                USING OPTLIST
                                             DDN-LIST
                                             HDNGLIST.
      B040-TERMINATION-EXIT.
          EXIT.
      C010-ERROR-RTN.
          MOVE NEXT-STEP(RSUB)         TO ERR-STEP.
          MOVE NEXT-PGM(RSUB)          TO ERR-PGM.
          MOVE RETURN-CODE             TO ERR-CC.
          DISPLAY BAD-COND-MSG         UPON SYSOUT.
          GOBACK.
      C010-ERROR-RTN-EXIT.
          EXIT.
      C020-DYNALLOC.
     *--------------------------------------------------------------*
     *    Create utility control file (SYSIN).                      *
     *--------------------------------------------------------------*
          MOVE NEXT-DSN (RSUB)         TO DYN-DSN.
          INSPECT DYN-DSN              REPLACING FIRST ' '
                                       BY ')'.
          CALL DYN-PGM                 USING ALLOC-DYNFILE.
          OPEN INPUT                   DYN-FILE
               OUTPUT                  CNTL-FILE.
          PERFORM                      UNTIL DYN-DONE
            READ DYN-FILE              INTO UTIL-REC
                                       AT END MOVE 'D' TO EOF-DYNFILE
                                       NOT AT END
                                       WRITE CNTL-REC FROM UTIL-REC
            END-READ
          END-PERFORM.
          CLOSE                        DYN-FILE, CNTL-FILE.
          MOVE 'R'                     TO EOF-DYNFILE.
      C020-DYNALLOC-EXIT.
          EXIT.
     *--------------------------------------------------------------*
     *    Physical end of program - HARNESS.                        *
     *--------------------------------------------------------------*
```

Using a Harness to control execution

One of the invoked programs

This is one of the programs used to test the harness. This program loads a VSAM KSDS and was modified to exploit Dynamic File Allocation using BPXWDYN, with the DDNAME and DSN associated with the KSDS being input via the KSDSDSNS file.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.                        KSDSLOD.
 AUTHOR.                            T.R.SAMBROOKS.
     INSTALLATION.
     DATE-WRITTEN.                  29TH AUG 2015.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*--------------------------------------------------------------*
* SAMPLE PROGRAM TO LOAD AN EMPTY VSAM KSDS DATA SET.          *
*--------------------------------------------------------------*
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT DSNS-IN                 ASSIGN TO UT-S-KSDSDSNS.
     SELECT ENGINES-IN              ASSIGN TO UT-S-INDD.
     SELECT KSDS-FILE               ASSIGN TO INDEXED-OUTDD
                                    ORGANIZATION IS INDEXED
                                    ACCESS IS DYNAMIC
                                    RECORD KEY IS LOCO-KEY
                                    FILE STATUS     IS FSTAT-CODE
                                                       VSAM-CODE.
 DATA DIVISION.
 FILE SECTION.
 FD  DSNS-IN                        RECORDING MODE IS F
                                    LABEL RECORDS ARE STANDARD
                                    BLOCK CONTAINS 0 RECORDS
                                    RECORD CONTAINS 80 CHARACTERS
                                    DATA RECORD IS DSN-REC.
 01  DSN-REC                        PIC X(80).
```

Using a Harness to control execution

```
FD  ENGINES-IN                  RECORDING MODE IS F
                                LABEL RECORDS ARE STANDARD
                                BLOCK CONTAINS 0 RECORDS
                                RECORD CONTAINS 80 CHARACTERS
                                DATA RECORD IS FB-LOCO-REC.
01  FB-LOCO-REC.
    COPY F100REC.
FD  KSDS-FILE                   RECORD CONTAINS 74 CHARACTERS
                                DATA RECORD IS KSDS-REC.
01  KSDS-REC.
    COPY F74REC.
WORKING-STORAGE SECTION.
01  WS-ADHOC-CONSTANTS.
    03  SUB                     PIC S9(4) COMP VALUE +1.
    03  DYN-PGM                 PIC X(8) VALUE 'BPXWDYN '.
    03  FSTAT-CODE              PIC XX.
    03  VSAM-CODE.
      05  R15-RETURN            PIC 99  COMP.
      05  VSAM-FUNCTION         PIC 9   COMP.
      05  VSAM-FEEDBACK         PIC 999 COMP.
    03  WS-EOF                  PIC X   VALUE 'M'.
      88  ALL-DONE              VALUE 'D'.
01  ALLOC-DYNFILE.
    03                          PIC S9(4) COMP VALUE +80.
    03                          PIC X(9) VALUE
       'ALLOC FI('.
    03  DYN-DDNA                PIC X(9) VALUE SPACES.
    03                          PIC X(17) VALUE
       ' SHR MSG(WTP) DA('.
    03  DYN-DSN                 PIC X(45) VALUE SPACES.
01  UNALLOC-DYNFILE.
    03                          PIC S9(4)  COMP VALUE  +27.
    03                          PIC X(8) VALUE
       'FREE FI('.
    03  DYN-DDNU                PIC X(8) VALUE SPACES.
    03                          PIC X(2) VALUE '  '.
    03                          PIC X(9) VALUE 'MSG(WTP) '.
PROCEDURE DIVISION.
```

Using a Harness to control execution

```
 A010-MAIN-PGM SECTION.
     PERFORM B010-INITIALIZE.
     PERFORM                     UNTIL ALL-DONE
       READ ENGINES-IN           AT END
                                    MOVE 'D' TO WS-EOF
            NOT AT END           MOVE FB-LOCO-KEY   TO LOCO-KEY
                                 MOVE FB-COY        TO COY
                                 MOVE FB-CAT-NO     TO CAT-NO
                                 MOVE FB-PRICE      TO PRICE
                                 MOVE FB-LOCO-DES   TO LOCO-DES
                               MOVE FB-LOCO-POWER TO LOCO-POWER
                               MOVE FB-LOCO-NAME  TO LOCO-NAME
                                 WRITE KSDS-REC
       END-READ
     END-PERFORM.
     PERFORM B020-TERMINATION.
 A010-MAIN-PGM-EOJ.
     GOBACK.
 B010-INITIALIZE.
     OPEN                        INPUT DSNS-IN.
     READ DSNS-IN                AT END CLOSE DSNS-IN.
     UNSTRING DSN-REC            DELIMITED BY '='
                                 INTO DYN-DDNA, DYN-DSN.
     INSPECT DYN-DDNA            REPLACING FIRST ' ' BY ')'.
     INSPECT DYN-DSN             REPLACING FIRST ' ' BY ')'.
     CALL DYN-PGM                USING ALLOC-DYNFILE.
     OPEN                        INPUT ENGINES-IN
                                 OUTPUT KSDS-FILE.
 B010-INITIALIZE-EXIT.
     EXIT.
 B020-TERMINATION.
       CLOSE                     ENGINES-IN
                                 KSDS-FILE.
       MOVE DYN-DDNA             TO DYN-DDNU.
       CALL DYN-PGM              USING UNALLOC-DYNFILE.
 B020B-TERMINATION-EXIT.
     EXIT.
*-------------------------------------------------------------*
* THIS IS BOTH THE LOGICAL AND PHYSICAL END OF - KSDSLOD      *
*-------------------------------------------------------------*
```

Using a Harness to control execution

Invoking JCL

This is the JCL used to test the harness. It accepts two parameters via the EXEC statement PARM field. The first 4-bytes indicate the type of run, which is separated from the 3-byte step name by a comma. As coded the program only caters for a 10 program schedule. The run types currently supported are ALL, run the entire schedule or ONLY, run just the one program. The SCHEDULE DD statement provides the schedule information as a series of 80-byte records. Each record contains the stepname (cols 1-3), program name (cols 5-12), an acceptable return code (cols 14-15) and for utility programs the DSN associated with SYSIN can be coded in columns 17 onwards.

```
//         EXPORT SYMLIST=STU
//         SET  STU=&SYSUID
//S0010    EXEC PGM=HARNESS,PARM='ALL ,S03/ALL31(OFF),STACK(,,BELOW)'
//STEPLIB  DD   DISP=SHR,DSN=&STU..LOAD.LIBRARY
//SYSOUT   DD   SYSOUT=*
//STEPMSGS DD   SYSOUT=*
//GENPRINT DD   DUMMY
//GENSYSIN DD   DUMMY
//SYSIN    DD   LRECL=80,RECFM=FB,BLKSIZE=0,
//             SPACE=(TRK,5)
//SYSPRINT DD   LRECL=125,RECFM=VBA,BLKSIZE=1254,
//             SPACE=(TRK,5),DISP=(MOD,PASS)
//SCHEDULE DD   *,SYMBOLS=JCLONLY
S01 IDCAMS   00 IDCAMS01 &STU..SOURCE.COB(IDCAMS01)
S02 KSDSLOD  00
S03 IDCAMS   00 IDCAMS03 &STU..SOURCE.COB(IDCAMS03)
S04 RRDSLOD  00
//KSDSDSNS DD   *,SYMBOLS=JCLONLY
OUTDD=&STU..KSDS.FILE
//INDD     DD   DISP=SHR,DSN=&STU..SOURCE.PLI(ENGINES)
//RRDSDSNS DD   *,SYMBOLS=JCLONLY
DIESEL=&STU..RRDS.DIESEL
STEAM=&STU..RRDS.STEAM
//MODELSIN DD   DISP=SHR,DSN=&STU..SOURCE.PLI(ENGINES)
```