```
!EMSDAC.HLP      *** 1981-12-07  EMS/EUN/MRH
!
!This document contains information designed for FORTRAN users who
!wish to use the EMSDAC subroutine package for creating digital sound
!data files.
!
!Contents:
!  EMSDAC - general description
!        Linking EMSDAC programs
!        Sound generation
!              frames of sound
!              generator construction
!              channel distribution
!              frequency
!              intensity
!              phase
!              FM
!              distortion
!        Files
!              names
!              structure
!              data format
!        Subroutines
!              calling procedures
!              no-parameter flag
!              detailed description of calls
!        Efficiency
!        Error handling
!        Compatibility with PDP/15-XVM programs
!        Global names
!        Advanced programming facilities
!              structure
!              subroutine details at lower program levels:
!                DAGN-, DASM-, DARC-
!              updating EMSDAC
!
```

```
!                    ***************************
!                    *** GENERAL DESCRIPTION ***
!                    ***************************
```

1 EMSDAC

!

EMSDAC is a group of programs that allows composers to create disk sound files containing data for the D/A converter. All routines in EMSDAC can be called from user FORTRAN programs.

Sound is created by up to 256 software sine-wave generators that can be connected to each other to produce more or less complex frequency modulation. The sound can be distributed on one to four channels through 256 channel distributors.

Users familiar with the EMSDEV program package available on the PDP-15/XVM will find many similarities here: one of the prime considerations in programming EMSDAC has been to facilitate the transfer of old programs to the VAX-11.

2 Linking

!

To gain access to EMSDAC routines, users must load two obect library files together with their own main programs and subroutines:

* DACLIB, which contains all EMSDAC subroutines
* MESLIB, which contains routines called by EMSDAC to write error messages

Assuming that the user has a main program called MAIN and a subroutine called SUB, EMSDAC can be loaded with the following command:

```
  $ LINK MAIN,SUB,-
      [EMSLIB]DACLIB/LIBRARY/INCLUDE=(DAGND,DASMD,DARCD),-
      [EMSLIB]MESLIB/LIBRARY/INCLUDE=MESSD
```

The qualifier /INCLUDE=MESSD may be omitted if the user does not wish EMSDAC messages to be displayed when the program is run.

!

```
!                       ************************
!                       *** SOUND GENERATION ***
!                       ************************
```

2 Sound

A disk sound file is opened with a call to TAPE. Music is then created as a series of "sound slices", each with a duration defined by TIME. Before each call to TIME, the sound is described with any number of changes to generator and amplifier parameters. The sound file is closed with ENDPLY, after which further files can be opened with TAPE. Not more than one file may be open at any given time.

Sound is created with 256 software sine-wave generators, each of which is connected either to any one of 256 channel distributors, or to the frequency input of another sine-wave generator.

The following system parameters are set to their default values every time a sound file is opened with TAPE. They can be altered with calls to the routines named below:

parameter	default	possible range	set by
sampling rate	50 kHz	25000 - 50000 Hz	SRATE
number of channels	1	1 - 4	CHANS

!

3 frames

!

Digital sound data is calculated internally in "frames" of 10 ms duration. Device parameters can be set only at the beginning of each frame; values are then interpolated within the frame between the previous settings and the new ones.

```
        Pi input  P2 input            P3 input
          here      here                here
           V         V                   V
           . . . . . . . . . . . . . . . . . . . . .
           .      P1 *         .         .      P3 *
           .        *. *       .         .       * .
           .       * .   *     .         .     *   .
           .      *  .     *   .         .   *     .
           .     *   .       * .         . *       .
           .    *    .         * * * * * *         .
           .   *     .      P2 .         .         .
           .  *      .         .         .         .
           . *       .         .         .         .
           .*        .         .         .         .
           * . . . . . . . . . . . . . . . . . . . .
           0        10        20        30        40 ms

            frame 1   frame 2   frame 3   frame 4   etc
```

The practical consequences of this are:

* generator frequencies and intensities and amplifier intensities all have onset times of 10 milliseconds: it always takes one frame for parameters to reach any new values set on them.

* in calls to TIME, it is safest to specify durations that are exact multiples of 10; the user's "sound slices" will then coincide with the device frames. When durations other than multiples of 10 are given, individual events may be displaced by up to 10 milliseconds.

The following diagram shows what actually happens to frequency and intensity when a generator is instructed to play a note at 100 Hz and 0 dB for a duration of 20 milliseconds, assuming that its previous settings were 0 Hz and -100 dB. (Note that intensity is in fact interpolated such that change is linear on an amplitude scale.)

```
    Hz                                 dB
    .................                 ..................
    100       ******                     0       ******
             *                                  *
            *                                  *
           *                                  *
          *                                  *
      0  *                            -100  *
    .................                 ..................
         0   10   20 ms                     0   10   20 ms
```

Frame duration is set by default to 10 ms every time a new sound file is opened. Other durations can be specified with calls to CSTEP. (See under the heading "Subroutines CSTEP".)

!

3 **generators**
!

The software sine-wave generators have been constructed so as to allow the output of each generator to be redirected and added to the frequency input of any other generator, thus producing frequency modulation.

```
***************************************************
*        freq           ampl       alternative    *
*          |             |           outputs      *
*          v             v         .............  *
* (--->) ...................       :  channel  :  *
* output :   :         :   :    -> :distributor:  *
*  from  :   :         :   :   ^   :...........:  *
*  other : + :   SIN   : x :-->|                  *
*  gens  :   :         :   :   v   .............  *
* (--->) :   :         :   :    -> :   other   :  *
* (--->) :...:.........:...:       : generator :  *
*                                  :...........:  *
*                                                 *
*         Sound generator construction            *
***************************************************
```

- a frequency is input, optionally added to the output of one or several other generators

- sine values are calculated using the FORTRAN library function SIN

- an amplitude is input and multiplied by the sine value to produce an output signal

- the output signal is added:
  - EITHER to one of the channel distributors
  - OR to the frequency input of another generator, in which case it modulates the frequency of the other generator.

For information on how to redirect generator outputs, see under the heading "Subroutines: FMCON".
!

3 channel distribution

!

Sound is distributed on up to 4 channels (CHA) by 256 channel distributors (CD). Any number of generators can be connected to any CD.

First the outputs of all generators that are connected to a given CD are added together. The sum is then multiplied by the CD amplitudes for those channels that are in use, and added to the final channel outputs, together with the output of all other CDs.

```
*******************************************
*                               channel   *
*  generator       CD           outputs   *
*  output(s)   amplitudes       .......   *
*     ! !           !           :  +  :   *
*     v v           v    ---->  :  1  :   *
*    ..................  /      :.....:   *
*    :   :        : x :/        .......   *
*    :   :        :...:         :  +  :   *
*    :   :        : x :------>  :  2  :   *
*    : + :   CD   :...:         :.....:   *
*    :   :        : x :         .......   *
*    :   :        :...:------>  :  +  :   *
*    :   :        : x :         :  3  :   *
*    :...:........:...:\        :.....:   *
*                       \       .......   *
*                        ---->  :  +  :   *
*                               :  4  :   *
*                               :.....:   *
*    Channel Distributor construction     *
*******************************************
```

See "Subroutines CDQ" for information on controlling CD amplitudes.
See "Subroutines FMCON" for information on connecting generators to CDs.

!

frequency and intensity

3 frequency

Sound generators have a frequency range of 0 - r/2 Hz, where r is the sampling rate in Hz. The default range is 0 - 25000 Hz.

See "Subroutines SRATE" for information on altering the sampling rate.

3 intensity

Intensities are specified on a 1/4 dB scale between 0 and 560, where 0 is equivalent to -100 dB and 560 is equivalent to +40 dB. It should be noted that, although intensities of up to +40 are allowed on individual generators, the total intensity of all generators being played should not exceed 0 dB if distortion is to be avoided. The following table gives a guide to the maximum intensities that can safely be used when different numbers of generators are playing. The column on the right shows how amplitudes are represented internally, on a linear scale from 0.0 to 1.0.

	max. intensity on each generator		
number of			internal
generators	dB	1/4 dB	amplitude
1	0	400	1.0
2	-6	376	0.5
4	-12	352	0.25
8	-18	328	0.125
16	-24	304	0.0625

Here it is assumed that intensity on channel output amplifiers is 400 (0 dB). The same effect can also be achieved by setting 400 on all the generators and reducing the channel output intensities to the figure in the 1/4 dB column; for example, if 16 generators are all put to 400, a channel intensity greater than 304 may result in sound distortion.

Note that intensities of -100 dB (zero on the 1/4 dB scale) are represented internally as amplitude zero (0.0), and not as their logarithmic equivalent 0.00001.

3 phase

The current phase of each generator is defined as a real number in the range 0.0 to 1.0; this range describes one cycle of a sine-wave.

The phase values of all generators are automatically put to zero when a sound file is opened. From then until the file is closed they are updated every sample for all generators that are active. (For a definition of "active generators", see under the heading "Efficiency".)

Users may set values on generator phases with calls to FGP. The values he gives need not be in the range 0 - 1: they are adjusted internally with the Fortran function MOD. For example:

user value	actual value put on generator
98.32	0.32
1.9	0.9
-1.9	0.1
-98.32	0.68

3 FM

!

An FM generator is created by connecting the output of one or more sine-wave generators to the input of another. Parameters on the modulator are then interpreted as modulation frequency and modulation index, rather than carrier frequency and intensity.

The modulated generator may in turn be connected to the input of another generator, and so on, thus forming more or less complex FM generators. When a sound file is opened, 16 pairs of sine-wave generators are by default connected to make 16 FM generators, while the remaining generators are treated as simple sine-wave oscillators. See under the headings "Subroutines TAPE" and "Subroutines FMCON" for more information on default connections.

There are two ways of setting parameter values on FM generators: with calls to FGQ and FM. These are described in more detail under the heading "Subroutines", but the main differences are as follows:

* FM can only be used for the 16 default FM generators, which are in fact identical with sine-wave generators 25 - 56. FGQ can be used for all generators, including those that can be set with FM.

* FM has floating-point arguments; FGQ has integer arguments.

* In FM, modulation index is defined on a linear scale between 0 and 100. In FGQ, modulation index is defined on a logarithmic scale between 0 and 560. The relation between the two scales is shown by this table:

FM (linear)	FGQ (logarithmic)
0.0	
0.00001	0
0.1	320
1.0	400
10.0	480
100.0	560

!

3 distortion

Signal distortion occurs when the final amplitude on any output channel is greater that 1.0. EMSDAC advises the user of the occurrence and location of distortion by means of messages on unit 6, e.g.:

```
AMP ERR IN FRAME STARTING SAMPLE     294501
```

though calculation continues even after the discovery of the error (i.e. this is not a fatal error). At the same time a file is created on unit 80 containing all generator parameter values at the beginning and end of the offending frame. This file, called FOR080.DAT, can be inspected at the end of the run; here is an example of what a "distortion file" can look like:

```
***** AMP ERR IN FRAME STARTING SAMPLE     294501
                            FREQUENCY    AMPL/FREQ DEVIATION
GENERATOR  CONNECTED TO   START    FINAL    START    FINAL    PHASE
----------------------------------------------------------------------
     1         CD  1     2100.00  2100.00  0.00000  1.00000  3.273239
    14         CD  1      148.00   148.00  1.00000  1.00000  0.075344
    18         FG 19        0.00    86.00 43.00000 43.00000  6.282276
    19         CD  1       86.00    86.00  0.00000  0.50000  4.214465
CHANNEL DISTRIBUTOR   CHANNEL   START AMPL   FINAL AMPL
--------------------------------------------------------
            1            1        1.00000      1.00000
      4: TOTAL NUMBER OF ACTIVE GENERATORS
      1: TOTAL NUMBER OF ACTIVE CHANNEL DISTRIBUTORS
      1: NUMBER OF CHANNELS
```

Note the way in which the following parameters are represented here:

- amplitude: 0 - 1
- modulation index: frequency deviation, calculated by the formula d = c * i, where c is the carrier frequency, i is the modulation index, and d is the resulting frequency deviation. In the example above it can be seen that generator 18, which has been connected so as to modulate generator 19, must have a modulation index of 0.5
- phase: 0 - 6.283185308

The signal that results when distortion occurs looks something like this (... = intended signal, xxx = actual signal):

```
                        ...                       ...
                       .   .                     .   .
+1 ------------------ x --- x ----------------- x --- x ---------------
                     x       x                 x       x
                    x         x               x         x
     x             x           x             x           x
      x           x             x           x             x
       x         x               x         x               x         x
        x       x                 x       x                 x       x
-1 ----- xxxxxxx ----- xxxxx ----- xxxxxxx ----- xxxxx ----- xxxxxxx --
          .   .                     .   .                     .
           ...                       ...                       ...
```

```
!                 *****************************************
!                 *** SOUND FILES - names and structure ***
!                 *****************************************
2      Files
!      -----
```

EMSDAC uses standard FORTRAN I/O statements to create files; their format precludes their being copied to magnetic tape with the Command Language instruction COPY.

```
3         names
!         -----
```

EMSDAC sound files are given names of type TAPEnn.DAC, where nn is a number from 01 to 99. See also under the heading "Subroutines TAPE".

```
3         structure
!         ---------
```

Files are created, written and closed with the standard FORTRAN statements OPEN, WRITE and CLOSE. Their structural organization and the type of access permitted is shown by the OPEN statement:

```
          OPEN (
     -          UNIT = lun,
     -          FILE = name,
     -          ACCESS = 'DIRECT',
     -          ORGANIZATION = 'RELATIVE',
     -          FORM = 'UNFORMATTED',
     -          STATUS = 'UNKNOWN',
     -          RECL = 2048)
```

where:

'lun' is a logical unit number (81 - 83) set internally by EMSDAC; these units are normally assigned to the user's default directory

'name' is the file-name, as described above

RECL defines a record length of 2048 longwords (4096 16-bit words)

STATUS UNKNOWN means that if a file of the given name already exists, it is opened and written over: the system does not create a new file of the same name with a higher version number.

Files are written to and read from with:

```
       INTEGER*2 BUFFER (4096)
                                      where:
       .
       WRITE (lun, REC=r) BUFFER        'lun' is a logical unit number, as
       .                                      above
       .                                'r' is a record number within the
       READ (lun, REC=r) BUFFER               file
```

For more information on file structures, refer to the VAX-11 FORTRAN Users Guide.

!

3 data format

As wave-form data is generated, samples are stored on disk in records of 4096 data items. The actual number of samples per record depends on how many channels the sound is to be distributed on: when there are n channels, n data items are created for each sample. For four-channel sound, for instance, information is arranged as follows:

position in record	sample number	channel number
1	1	1
2		2
3		3
4		4
5	2	1
6		2
7		3
8		4
9	3	1
etc.		

For more information, see under "Advanced: DASM-"

```
!             ************************************************
!             *** SUBROUTINE NAMES AND CALLING PROCEDURES ***
!             ************************************************
2    Subroutines
!    -----------
```

EMSDAC contains the following routines:

TAPE	opens a sound file on disk
FMCON	makes (dis-)connections, both FM & to channel distributors
CDQ	sets intensities on channel distributors
CLEMS	clears all device parameters and connections
CHANS	determines the number of sound channels
CSTEP	sets calculation step time (frame duration)
FGP	sets generator phases
FGQ	sets frequency and intensity on sine-wave generators
FM	sets parameters on pairs of connected sine-wave generators
SRATE	sets sampling rate
TIME	defines the delay before the next parameter changes are to take effect
ENDPLY	closes the sound file

The following routines exist only because they exist on the PDP-15/XVM system; new programs should use FMCON and CDQ:

CONNEC	connects generators to channel distributors
DISCON	disconnects generators
AMPLQ	sets intensities on the outputs of channel distributor 1

See also the section headed "Compatibility".

!

calling procedures and the no-parameter flag

3 calling procedures

All EMSDAC programs are called as INTEGER FUNCTION subprograms that return a function value which describes the success or failure of the operation. For example:

```
          I = FGQ (1, 220, 360, 0)
```

puts parameter values on generator 1 and returns a code to the variable 'I'. In general, negative codes indicate error conditions, while positive codes indicate successful operation.

It is also possible to invoke EMSDAC subroutines with a simple CALL, in which case no status codes are returned. For example:

```
          CALL FGQ (1, 220, 360, 0)
```

N.B. * data-type declarations must be written at the head of programs that call EMSDAC routines as functions. For example:

```
          INTEGER AMPLQ, CDQ, CLEMS, CONNEC, DISCON, ENDPLY
          INTEGER FGQ, FM, FMCON, TAPE, TIME
```

* when TIME is invoked with CALL TIME (i.e. not as a function), it must be declared thus at the head of the program unit, to avoid confusion with the system routine of the same name:

```
          EXTERNAL TIME
```

3 no-parameter flag

In AMPLQ, CDQ, FGP, FGQ and FM, the user may indicate parameters that are to remain unaltered by giving them the special value -131070 (or -131070.0 in the case of FGP and FM, which have real-number arguments). Judicious use of this facility will speed up program execution, particularly in conjunction with generator intensities. For example:

```
               NOP = -131070
               I = FGQ (2, NOP, 328, NOP)
```

Here generator 2 receives a new intensity, but frequency and wave form remain unchanged. Similarly:

```
               ANOP = -131070.0
               I = FM (10, 235.5, ANOP, 127.75, ANOP)
```

puts new modulation and carrier frequencies on FM 10, while the modulation index and intensity are unaltered.

```
************************************************************
*** DESCRIPTION OF SUBROUTINES IN ALPHABETICAL ORDER ***
************************************************************
```

3 AMPLQ

Integer function subprogram that sets intensity on one of the four channel outputs of channel distributor #1. These are numbered from 19 to 22 for compatibility with programs written to run on the PDP-15/XVM.

```
        AMPLQ (AMPNR, INTENS)

   AMPNR  - integer - amplifier number (19 - 22)
   INTENS - integer - intensity in 1/4 dB (0 - 480)
   Function value - INTEGER*4:
               1 OK
               2 OK, but INTENS had the value -131070, so no intensity
                 was set
             -41 illegal amplifier number . Message written on
                 unit 6. Program returns without doing anything
             -42 illegal amplifier intensity. Error message
                 output, and intensity set to nearest legal
                 limit (0 or 480).
```

External call: MESS

3 CDQ

!

Integer function subprogram that sets intensity on channel distributor amplifiers

```
       CDQ (CDNR, CHAN, INTENS)

  CDNR - INTEGER*4 - channel distributor number (1 - 256)
  CHAN - INTEGER*4 - output channel (1 - 4)
  INTENS - INTEGER*4 - intensity in 1/4 dB (0 - 480)
  Function value - INTEGER*4:
              1 OK
              2 OK, but INTENS had the value -131070, so no intensity
                was set
            -41 illegal CDNR or CHAN
            -42 illegal intensity
```

Note that, by default, sound is produced for channel output #1 only. Attempts to set amplitudes on non-existent outputs (2 - 4) are not flagged as errors.

For information on how to change the default number of outputs, see under the heading "Subroutines CHANS".

Error conditions: Message is written on unit 6, and amplitudes remain
                  unchanged

External call: MESS

!

3 CHANS

Integer function program that determines the number of channel outputs sound is to be distributed on. By default, there is one channel; this default condition is set every time TAPE is called.

CHANS should therefore be called after TAPE, but before TIME is called for the first time.

```
      CHANS (NCHA)

  NCHA - INTEGER*4 - number of channel outputs (1 - 4)
  Function value - INTEGER*4:
              1 OK
            -41 illegal number of channels.  Error message on unit 6,
                and no change made to current number of channels.
```

Note that there is a further restriction on the value that may be assigned to NCHA:

NCHA * KHZ * STEPTIME may not exceed 2000

where

NCHA	is the number of channel outputs
KHZ	is the sampling rate in kHz
STEPTIME	is the calculation time set by CSTEP

External calls: DAGNP, MESS

3 CLEMS

!

Integer function program that clears internal tables containing all:

- * generator frequencies, intensities and phases
- * channel distributor amplifier intensities
- * connection points
- * generator "active" flags (see under the heading "Efficiency")

```
      I = CLEMS ()      or      CALL CLEMS

  Function value - INTEGER*4:
              1 OK
```

No error conditions or external calls

!

3 CONNEC

!

Integer function subprogram that makes connections between generators and channel distributors.

```
        CONNEC (FROM, TO)

   FROM    - INTEGER*4 - generator number (1 - 256)
   TO      - INTEGER*4 - channel distributor number (1 - 256)
   Function value - INTEGER*4:
                 1 OK
               -51 illegal generator number
               -52 illegal channel distributor number
```

Error conditions: illegal argument values are flagged with a message
                  on unit 6, and no connection is made

External call: MESS

!

3 CSTEP

!

Integer function subprogram that determines the step time for generator calculations (frame duration). The default condition, set every time TAPE is called, is a step time of 10 milliseconds.

See under the heading "Sound frames" for more information.

```
        CSTEP (TIME)

  TIME - INTEGER*4 - duration in milliseconds of every calculation step
                          (1 - 40)
  Function value - INTEGER*4:
              1 OK
            -31 illegal step time.  Error message written on unit 6, and
                  no alteration made to the current step time.
```

Note that there is a further restriction on the value that may be assigned to TIME:

```
            NCHANS * KHZ * STEPTIME may not exceed 2000
where
    NCHANS      is the number of channel outputs
    KHZ         is the sampling rate in kHz
    STEPTIME    is the calculation time set by CSTEP
```

External calls: DAGNP, MESS

!

3 DISCON

Integer function subprogram that disconnects generators

DISCON (NR, DUMMY)

NR - INTEGER*4 - generator number (1 - 256)
DUMMY - INTEGER*4 - dummy value for compatibility with PDP/15-XVM
programs. Has no significance here.
Function value - INTEGER*4:
1 OK
-51 illegal generator number: message is displayed and no
disconnection is performed

External calls: none

3 ENDPLY
!

Integer function subprogram that closes the disk sound file that was opened with TAPE. Returns information about total music duration and number of calls to TIME since the latest call to TAPE.

```
        ENDPLY (TIMES, SECS, MSECS)

    TIMES - INTEGER*4 - gets the number of times TIME has been called
    SECS & MSECS - INTEGER*4 - get the music duration since the latest
                    call to TAPE.  SECS gets the seconds part of the
                    duration, while MSECS gets the milliseconds part.
    Function value - INTEGER*4:
                  1 OK
                -91 error: could be TAPE not called, or sound
                    file too large, or internal I/O error
```

Error conditions: the file is not closed; however, values are returned to arguments TIMES, SECS and MSECS, though they may be erroneous.
A message is written on unit 6.
External calls: DAGNE, MESS
!

3 FGP

!

Integer function subprogram that sets an absolute phase value in the range 0.0 to 1.0 on a specified generator.
N.B. At the moment this routine does nothing at all, except check the legality of argument NR.

```
        I = FGP (NR, PHASE)

  NR - INTEGER*4 - generator number (1 - 256)
  PHASE - REAL*4 - phase value.  Values outside the range (0.0,1.0)
         are adjusted internally with MOD.
  Function value - INTEGER*4:
              2 OK, but nothing done because PHASE had the value of the
                no-parameter flag: -131070.0
              1 OK
            -21 illegal generator number
```

External call: MESS

!

3 FGQ
!

Integer function subprogram that sets frequency and intensity on a specified frequency generator

```
        FGQ (NR, FREQ, INTENS, WF)


   NR     - INTEGER*4 - generator number (1 - 24)
   FREQ   - INTEGER*4 - frequency in Hz (0 - 15999)
   INTENS - INTEGER*4 - intensity in 1/4 dB (0 - 560), or modulation
             index on a logarithmic scale (400 = MI 1.0) if this
             generator modulates another one
             (see also under the heading "Sound generation: FM")
   WF     - INTEGER*4 - wave form (0 or 1) = sine wave: no other wave-
             forms are available
   Function value - INTEGER*4:
               1 OK
             -21 illegal generator number
             -22 illegal frequency
             -23 illegal intensity
             -24 illegal wave form
```

Error conditions:
If illegal generator number is found, the program writes an error message and returns immediately. If illegal parameter values are found, the parameter in question is automatically set to the nearest legal value, and a message is written.
For example:

```
             I = FGQ (12, 16324, -10, 1)
```

gives the same result as:

```
             I = FGQ (12, 15999, 0, 1)
```

except that an error message is written, and 'I' receives status code -22.

If any parameter (except NR) has the value -131070, the parameter remains unaltered, and there is no error code or message.

External call: MESS
!

3 FM

!

Integer function subprogram that sets modulation frequency, modulation index, carrier frequency, and intensity on a specified FM generator.

```
      FM (NR, MFREQ, MINDEX, CFREQ, INTENS)

   NR     - INTEGER*4 - generator number (1 - 16)
   MFREQ  - REAL*4 - modulation frequency in Hz (0 - 16383.75)
   MINDEX - REAL*4 - modulation index (0 - 100)
   CFREQ  - REAL*4 - carrier frequency in Hz (0 - 16383.75)
   INTENS - REAL*4 - intensity in dB (0 - 100)
   Function value - INTEGER*4:
                 1 OK
               -81 illegal FM number
               -82 illegal modulation frequency
               -83 illegal modulation index
               -84 illegal carrier frequency
               -85 illegal intensity
```

FM generators are pairs of interconnected sine-wave generators. For compatibility with PDP/15-XVM programs they are numbered as follows:

FM no	sine gen no
1	25,26
2	27,28
3	29,30
.	.
.	.
16	55,56

Error conditions:
If illegal FM number is found, the program returns without performing any action (except writing message). If illegal parameter values are found, the parameters in question are put to the nearest legal value.

If any parameter (except NR) has the value -131070.0, the parameter remains unaltered, and there is no error code or message.

External call: MESS

!

3 FMCON

Integer function subprogram that directs the output of frequency generators (i.e. (dis-)connects them to the inputs of other generators, or to channel distributors).

```
      FMCON (TYPE, FROM, TO)

  TYPE - INTEGER*4 - type of connection to be performed
                0: disconnect
                1: connect to another generator's frequency input
                2: connect to a channel distributor
  FROM - INTEGER*4 - number of the generator whose output is to be
                      directed (1 - 256)
  TO - INTEGER*4 - meaning depends on TYPE
             if TYPE=0: TO has no significance
                TYPE=1: TO is the number of the generator which is to
                        be modulated by FROM (must be greater than FROM)
                TYPE=2: TO is the number of the channel distributor to
                        which FROM is to be connected (1 - 256)
  Function value - INTEGER*4:
             1 OK
           -51 illegal TYPE
           -52 illegal connection point (FROM or TO)
```

Error conditions:
A message is written on unit 6, and no (dis-)connection is made.

External call: MESS

Example:
The diagram below shows the construction of an extremely complex FM generator which is connected to the same channel distributor as a simple sine-wave generator (no. 11). On the right are the calls that must be made in order to produce this construction.

```
.....     .....     .....                 ........
: 1 : --> : 2 : --> : 4 : --              .      .
:...:     :...:  !  :...:   !             .      .
                 ^          !             .      .  CALL FMCON (1,1,2)
         .....   !          V             .      .  CALL FMCON (1,2,4)
         : 3 : ->           !             .      .  CALL FMCON (1,3,4)
         :...:              !             .      .  CALL FMCON (1,4,10)
                            !             . CD#1 .  CALL FMCON (1,5,7)
          .....     .....   V  ......     .      .  CALL FMCON (1,6,8)
          : 6 : --> : 8 : ---> : 10 : --> .      .  CALL FMCON (1,7,8)
          :...:  !  :...:  !   :.....:    .      .  CALL FMCON (1,8,10)
                 ^         !              .      .  CALL FMCON (1,9,10)
.....    .....   !         ^              .      .  CALL FMCON (2,10,1)
: 5 : -> : 7 : ->  .....   !              .      .  CALL FMCON (2,11,1)
:...:    :...:     : 9 : ->    ......     .      .
                   :...:       : 11 : --> .      .
                               :.....:    ........
```

For a description of default generator connections, see under the heading "Subroutine TAPE".

3 SRATE

!

Integer function subprogram that defines the sampling rate at which data is to be written to a sound file. The default rate, set every time TAPE is called, is 50000 Hz; if another sampling rate is required, SRATE should be called once only immediately after TAPE.

```
        SRATE (RATE)

  RATE - INTEGER*4 - sampling rate in Hz (25000 - 50000)
  Function value - INTEGER*4:
              1 OK
            -32 illegal sampling rate.  message written on unit 6, and
                no change made to current sampling rate.
```

Note that there is a further restriction on the value that may be assigned to RATE:

```
            NCHANS * STEPTIME * RATE/1000 may not exceed 2000
where
    NCHANS      is the number of channel outputs
    RATE        is the sampling rate in Hz
    STEPTIME    is the calculation time set by CSTEP
```

External calls: DAGNP, MESS

!

3 TAPE
!

Integer function subprogram that opens a disk sound file for writing digital sound data.

```
      TAPE (FILENR)

   FILENR - INTEGER*4 - specifies a disk file number (1 - 99).
            The file is given the name TAPEnn.DAC, where nn is the
            same number as FILENR.
   Function value - INTEGER*4:
               1 OK
             -11 illegal FILENR
             -12 probably file already open, i.e. ENDPLY not done since
                 last TAPE.  But could be other internal errors.
```

This routine also:

- * clears generator frequencies & intensities and amplifier intensities
- * clears generator phases
- * makes default connections - all generators are connected to channel distributor #1, except generators 25 to 56, which are connected in pairs as simple FM generators:

```
                                  FG 25 -> FG 26 -> CD1
   FG 1...24    -> CD1            FG 27 -> FG 28 -> CD1
                                  .
   FG 57...256  -> CD1            .
                                  FG 53 -> FG 54 -> CD1
                                  FG 55 -> FG 56 -> CD1
```

Error conditions:

- an appropriate message is written on Unit 6
- file is not opened and device parameters are not cleared

External calls: DAGNI, DAGNP, MESS
!

3 TIME

!

Integer function subprogram that calculates digital sound data for a specified number of milliseconds. Calculations are based on all subroutine calls since the last call to TIME (or TAPE, if this is the first call to TIME), as well as all unchanged parameter information.

```
        TIME (MS)

   MS  - INTEGER*4 - duration in milliseconds of this sound slice
                      (0 - 32767)
   Function value - INTEGER*4:
                   1 OK
                 -71 illegal MS - duration is set automatically to the
                        nearest legal value (0 or 32767)
                 -72 probably file not open.  Might also be internal I/O
                        error.  Message is written, and control is returned
                        at once to calling program.
```

External calls: DAGNW, MESS

!

```
!                          **************************
!                          *** PROGRAM EFFICIENCY ***
!                          **************************
!
```

2 Efficiency

In order to speed up the calculation of sound data, EMSDAC works only with those generators and channel distributors that are actually being used.

A generator is considered to be active if:

* it is connected to a channel distributor or another generator
* its "active" flag is set: this is a flag which is set automatically whenever a generator receives values with the calls FGQ and FM; when TAPE is done, all generator flags are automatically put to "inactive" by default
* both frequency and intensity are non-zero

If all these conditions are true, sound data is calculated for the generator; if one or more conditions are not satisfied, the generator is ignored.

A channel distributor is considered to be active if at least one active generator is connected to it.

The user can assist in the speeding-up process by ensuring that generators which have been used but which are not needed any more are marked as "inactive". There are two ways of doing this:

* by setting both frequency and intensity to zero
* by disconnecting the generator with FMCON (0, ...): this should be done, however, only when intensity is zero, if clicks are to be avoided

For example:

```
      PARAMETER (NOP=-131070, ANOP=-131070.)
      .
      .
* PLAY NOTES ON GENERATOR #20 & FM #1
      CALL FGQ (20, 440, 380, NOP)
      CALL FM (1, 347., 1.5, 173.5, 71.)
      CALL TIME (100)

* TURN THE INTENSITIES OFF
      CALL FGQ (20, NOP, 0, NOP)
      CALL FM (1, ANOP, 0., ANOP, 0.)
      CALL TIME (10)

* TURN THE FREQUENCIES OFF                * OR, ALTERNATIVELY, DISCONNECT
      CALL FGQ (20, 0, NOP, NOP)                    CALL FMCON (0, 20, NOP)
      CALL FM (1, 0., ANOP, 0., ANOP)               CALL FMCON (0, 25, NOP)
      CALL TIME (10)                                CALL FMCON (0, 26, NOP)
                                                    CALL TIME (10)
!
```

```
!                             **********************
!                             *** ERROR HANDLING ***
!                             **********************
2      Errors
!      ------
```

EMSDAC routines report errors in two ways:

- by returning status codes (when invoked as function subprograms)
- by writing messages to unit 6. These messages are of the form:

```
     ***          'S': PROGRAM UNIT 'P'
                  'V': TEXT DESCRIBING ERROR
```

where 'V' is the offending value that caused the error;
'P' is the name of the program in which the error was found;
'S' is the status code within that program.

For example:

```
                 I = FGQ (0, 100, 300, 0)
```

results in the message:

```
     ***      -21: PROGRAM UNIT FGQ
                0: ILLEGAL FG NUMBER
```

The offending value was 0, and -21 is the status code returned by program FGQ.

N.B.
Occasionally errors are reported by lower program levels. This may happen, for instance, when excessive amplitude levels cause distortion, or when an attempt is made to write a sound file without first opening one with TAPE.

The user who wishes to know what happens at lower program levels may study the information under the heading "Advanced".

Alternatively, it is possible to disenable the display of messages with the routine MESSP. More information can be obtained by studying the documentation file EMSMESS.HLP, or by giving the command:

```
      $ HELP EMSMESS
```

!

3 codes

For compatibility with programs written to run on the PDP-15/XVM, EMSDAC routines return the following codes as function values.

code	meaning	routines
2	no error, but nothing done because of the no-parameter flag NOP	AMPLQ, CDQ, FGP
1	operation performed successfully	all routines
-11	illegal disk file number	TAPE
-12	error on opening file	TAPE
-21	illegal generator number	FGQ, FGP
-22	illegal generator frequency	FGQ
-23	illegal generator intensity	FGQ
-24	illegal generator wave-form	FGQ
-31	illegal step time (frame-duration)	CSTEP
-32	illegal sampling rate	SRATE
-41	illegal amplifier number/number of channels	AMPLQ, CDQ, CHANS
-42	illegal amplifier intensity	AMPLQ, CDQ
-51	illegal generator number as connection point OR illegal device number (generator or CD)	CONNEC, DISCON FMCON
-52	illegal CD number as connection point	CONNEC
-53	illegal connection type	FMCON
-71	illegal time	TIME
-72	error on writing to file	TIME
-81	illegal FM number	FM
-82	illegal modulation frequency	FM
-83	illegal modulation index	FM
-84	illegal frequency on FM generator	FM
-85	illegal intensity on FM generator	FM
-91	error on closing file	ENDPLY

```
!          ********************************************
!          *** COMPATIBILITY WITH PDP/15-XVM PROGRAMS ***
!          ********************************************
```

2 Compatibility

!

The names, calling procedures and operations of EMSDAC routines correspond closely to those of the EMSDEV package in use on the studio's PDP-15/XVM. But note the following differences:

- * new studio devices
  - 256 interconnectable sine-wave generators
  - 256 channel distributors
- * missing studio devices
  - there are no ring-modulators, amplitude-modulators, frequency filters, frequency shifters, noise generators, or reverberation units
- * new routines
  - CDQ, CHANS, CSTEP, FGP, FMCON and SRATE
- * missing routines
  - FF, FFQ, FG and AMPL
- * alterations made to old routines
  - AMPLQ, CONNEC, DISCON: the meanings of the arguments to these routines are not the same as in the PDP/15-XVM system, because of changes made in the physical characteristics of the studio
  - FGQ: frequency and intensity range have been changed
  - generator intensities are NOT put automatically to zero every time TIME is called; they are cleared only when specific calls to FGQ, FM or CLEMS are issued

For more details see under "Sound generation" and "Subroutines".

!

!
****
!
*** GLOBAL NAMES USED IN EMSDAC ***
!
****

2 Globals

!

The following are the names of the subroutines, functions, common blocks and block-data programs used in the EMSDAC program package. Users may not use any of these names for their own programs or common blocks.

AMPLQ	P	O
AMPTAB	CB	O
BLDA	P	A-2
CDQ	P	O
CHANS	p	O
CONNEC	P	O
CONTAB	CB	O
CSTEP	p	O
DABL	P	A-2
DABLBK	CB	A-2
DADAT	P	A-2
DAGNBK	CB	-1
DAGND	BD	-1
DAGNE	P	-1
DAGNI	P	-1
DAGNP	P	-1
DAGNW	P	-1
DARCBK	CB	-3
DARCD	BD	-3
DARCE	P	-3
DARCI	P	-3
DARCR	P	-3
DARCW	P	-3
DASMBK	CB	-2
DASMD	BD	-2
DASME	P	-2
DASMI	P	-2
DASMW	P	-2
DISCON	P	O
DKFIL	CB	-2
~~DUMPAR~~	~~CB~~	~~-1~~
ENDPLY	P	O
FGCH	P	A-1
FGP	P	O
FGQ	P	O
FILBLK	CB	-2
FILNAM	CB	-1
FLGTAB	CB	O
FM	P	O
FMCON	P	O
GENTAB	CB	O
MESBLK	CB	A
MESS	P	A
MESSD	BD	A
MESSP	P	A
PHATAB	CB	O
SRATE	P	O
TAPE	P	O
TIMBLK	CB	O
TIME	P	O
USEPAR	CB	-1

P = PROCEDURE (subroutine or function)
CB = COMMON BLOCK
BD = BLOCK DATA

The numbers show the program levels at which the programs are used. 'A' beside the level indicator means that the routine is an auxiliary program called by programs at this level. 'A' without a level indicator marks an auxiliary program available to routines at all levels.

For additional information on program levels, see under "Advanced".

!

****

*** ADVANCED: WORKING AT LOWER PROGRAM LEVELS ***

****

So far, this document has given information intended primarily for composers who wish to work at the top level of EMSDAC. In effect, the subroutines described above provide an interface between the composer and the calculation of digital sound data.

What follows is concerned with lower program levels, and is therefore of interest only to programmers who wish to modify or replace existing routines or add new ones.

2 Advanced

!

EMSDAC is constructed on four program levels, 0 to -3. Level 0 is highest, i.e. lies nearest the composer, and -3 is closest to machine operations.

The prime considerations in structuring EMSDAC have been:

* to make it as easy as possible to add new routines that manipulate common block data - an example of this is given under the heading "Advanced Update".

* to make it as easy as possible to replace a complete program level. It is envisaged, for example, that it may at some time be desirable to replace the whole of level 0, which is a relic of the hybrid system with hardware analog sound generators controlled by a PDP/15-XVM. Through careful study of the input required by programs at level -1, the programmer should have little difficulty in constructing a software interface that is better suited to the facilities available in a fully digital system. An example of the replacement of a whole level can be found in EMSAP, the version of EMSDAC that makes use of the AP120 for generator calculation: here the whole of level -1 has been replaced in order to accommodate the differences of approach required in the use of an Array Processor, while all other routines are identical with those in EMSDAC.

3 calling procedures

!

All programs at level -1 and below are called as subroutines that return status codes as one of their arguments. They cannot be called as INTEGER FUNCTION subprograms. For example:

```
        CALL DAGNI (FNAME, STATUS)
```

Here the integer variable STATUS gets a positive value if the operation is successful, and a negative value if it fails for any reason.

!

3 structure

!

Each level consists of:

* COMMON BLOCKS containing data that the level works with - these should be accessed only by programs at the level in question, and not by programs at other levels

* one BLOCK DATA program that initializes all the common blocks at the level in question

* optionally, one or more routines that manipulate data in common blocks, e.g. put values in them, or convert data from one format to another; for example, FGQ, CONNEC and so on, at level 0

* optionally, one or more auxiliary routines that are called by programs at the level in question to perform data manipulation; for example, DABL and BLDA which are called by level -2 programs to convert data between floating-point and 16-bit integer formats

* routines that control the flow of data: they initialize data-flow, send data to lower levels, fetch data from lower levels, and close data-flow. In general, these routines are called in series: thus in the table below, TAPE calls DAGNI which calls DASMI which calls DARCI; TIME calls DAGNW which calls DASMW which calls DARCW; and so on. Some routines, however, call two lower level routines: DAGNE, for instance, calls both DASMW and DASME. gäller inte längre

The following tables describe what happens at each program level: what data each level expects as input, what it does with the data, and which program units are involved in the process.

!

Input of data to EMSDAC program levels

level 0 TAPE	level -1 DAGN-	level -2 DASM-	level -3 DARC-
frequency, amplitude, connections for individual generators & amplifiers. intensity on logarithmic dB scale	tables of all generator & amplifier ampls, freqs, connecs, "active"-flags. linear amplitude (0.0 to 1.0)	digital sound data in the range -1.0 to +1.0	digital sound data in an array of 4096 16-bit integers - one disk file record
duration in ms ("time-slices")	sample no. at end of previous event	sample no. at start of event	file record no.

program operation at each level, and names of data-flow routines

level	operation	init	send	fetch	close
0	converts intensity (db) to amplitude, stores freq, amplitude & connec data, sets generator "active"-flags, converts duration to sample number	TAPE	TIME	-	ENDPLY
-1	converts generator & amplifier parameters to floating-point digital sound data, keeps track of generator phases	DAGNI	DAGNW	-	DAGNE
-2	converts sound data from floating-point to 16-bit integer format	DASMI	DASMW	DASMR	DASME
-3	performs disk I/O with Fortran OPEN, WRITE, READ, and CLOSE	DARCI	DARCW	DARCR	DARCE

Common block manipulation

level	names of common blocks	routines that manipulate common blocks	block data
0	AMPTAB, CONTAB, FLGTAB, GENTAB, TIMBLK	AMPLQ, CDQ, CHANS, CLEMS, CSTEP, DISCON, FGP, FGQ, FM, FMCON, SRATE	
-1	DAGNBK, ~~DUMPAR~~, FILNAM, PHBLK, USEPAR	DAGNP, FGCH*	DAGND
-2	DASMBK, DKFIL, FILBLK, DABLBK*	BLDA*, DABL*, DADAT*	DASMD
-3	DARCBK	-	DARCD

names marked with * are auxiliary routines (or their common blocks) called by programs at these levels to perform manipulation of common blocks

!

3   ramps
!

We have already seen how data is organized in 10 ms frames, within which ramps are generated for frequencies and amplitudes. It should be noted that these ramps begin immediately before the first sample in the frame and reach their final values on the last sample. If, for example, a frequency envelope from 0 to 1000 Hz is generated in the first frame in a file (samples 1 to 500), the value calculated for sample 1 is 2 Hz, and the value for sample 500 is 1000 Hz. If the frequency is then to go down to 500 Hz by the end of frame 2, the value calculated for sample 501 is:

1000 + ((500 - 1000) / 500) = 999 Hz.

```
          1000 input          500 input
         Hz   V                  V
        1000  . . . . . . . . *  . . . . . . . . .
              .              *.  *               .
              .             * .  . *             .
              .            *  .  .   *           .
              .           *   .  .     *         .
              .          *    .  .       *       .
              .         *     .  .         *     .
              .        *      .  .           *   .
              .       *       .  .             * .
         500  .      *        .  .               *
              .     *         .  .               .
              .    *          .  .               .
              .   *           .  .               .
              .  *            .  .               .
              . *   frame 1   .  .     frame 2   .
              .*              .  .               .
              *               .  .               .
         0 (*). . . . . . . . .  . . . . . . . . .
              1             500  501           1000 samples
```

It can be seen that ramps start, in effect, from the value at the end of the previous frame, or, in the case of the first frame, from an imaginary sample #0.
!

```
!              ******************************************
!              *** SUBROUTINES IN ALPHABETICAL ORDER ***
!              ******************************************
!
```

3 BLDA

level -2

Called by programs at level -2 to convert sampling data in floating-point format between -1.0 and +1.0 to integer*2 (16-bit) format suitable for D/A conversion.

```
        CALL BLDA (ECOUNT, SOURCE, DESTIN, STATUS)

   ECOUNT - INTEGER*4 - number of array elements to be converted
   SOURCE - REAL*4 ARRAY - contains data between -1.0 and +1.0
   DESTIN - INTEGER*2 ARRAY - receives converted data
   STATUS - INTEGER*4 - gets:
                   1 OK
                  -1 illegal ECOUNT (must not be <0)
                  -2 amplitude errors found (data outside limits
                       -1.000015 and +1.000015)
```

Data is converted according to the following pattern:

```
                 1.0      <--->   +32767
                 0.0      <--->      0
                -1.0      <--->   -32767
          outside range   <--->   -32768
       (approx. -1.00003)
```

See DADAT for information on converting data which lies between limits other than -1.0 and +1.0

Error conditions: On illegal ECOUNT, a message is written on unit 6, and no conversion is performed. When amplitude errors are found, a special value (-32768) is put in the destination array instead of the illegal value. When all the data has been converted, a message is written stating how many data items have been flagged in this way.

External call: MESS

See DASMW for example of calling procedure

!

DABL

level -2

Converts sampling data in integer*2 (16-bit) format to real numbers in the range -1.0 to +1.0.

```
      CALL DABL (ECOUNT, SOURCE, DESTIN, STATUS)

  ECOUNT - INTEGER*4 - number of data items to be converted
  SOURCE - INTEGER*2 ARRAY - contains 16-bit data
  DESTIN - REAL*4 ARRAY - receives real numbers after conversion
  STATUS - INTEGER*4 - gets status code:
                   1 OK
                  -1 illegal ECOUNT (must not be <0)
                  -2 amplitude errors found (i.e. error flag -32768, set
                     in BLDA q.v.)
```

Error conditions: Illegal ECOUNT is flagged with a message on unit 6, and no conversion is performed. Amplitude errors are flagged when the conversion is complete - this is not a fatal error.

External call: MESS

For information on converting data to ranges other than -1.0 to +1.0, see DADAT.
For example of calling procedure, see DASMR.

3 DADAT
! ‾‾‾‾‾

level -2

Sets the multiplication and addition factors used by DABL and BLDA in converting D/A sound data. These factors provide the user with the means of controlling the overall amplitude of sound data and of scaling data which is not in the standard ranges -1.0 to +1.0 or -32767 to +32767.

```
      CALL DADAT (MUL, ADD, STATUS)
```

```
  MUL - REAL*4 - multiplication factor (default value: 32767.0)
  ADD - REAL*4 - addition factor (default value: 0.0)
  STATUS - INTEGER*4 - gets:
                1 OK
```

The formulae used for data conversion are as follows:

```
  * BLDA (conversion from real to 16-bit)         i = (r * mul) + add
  * DABL (from 16-bit to real)                    r = (i - add) / mul
```

where i is a 16-bit integer (range -32767 to +32767)
and   r is a real number (usually in the range -1.0 to +1.0)

No error conditions or external calls
!

```
3          DAGNE
!          -----
                                                         level -1

Closes the current sound file.

       CALL DAGNE (STATUS)

   STATUS - INTEGER*4 - gets:
                 1 OK
                -1 I/O error, or no file open (error in DASME)

External calls: DASME, MESS

See ENDPLY for example of calling procedure
!
```

3 DAGNI

level -1

Opens a disk file for the creation of digital sound data

```
        CALL DAGNI (NAME, STATUS)

   NAME - CHARACTER - file name (no extension)
   STATUS - INTEGER*4 - gets:
                  3 OK, named file already open
                  2 OK, existing file opened
                  1 OK, new file opened
                 -1 error in call to DASMI (I/O error): the file is not
                    opened
```

External calls: DASMI, MESS

For example of calling procedure, see TAPE.

3 DAGNP
!

level -1

Subroutine that puts values on EMSDAC system parameters. Care must be exercised in the use of this routine if strange results are to be avoided: it should be called after DAGNI, and before any call to DAGNW, though no checks are made to ensure that timing is right. All parameters that can be set here are put to their default values in DAGNI.

```
      CALL DAGNP (PARAM, VALUE, STATUS)

   PARAM - CHARACTER - text string specifying which parameter is to
            receive a value.  Legal strings are:
       'SAMPLING RATE' - sampling rate in Hz (25000-50000) default 50000
       'FRAME DURATION' - duration in ms of sound frames (>0) -"-   10
       'CHANNELS' - number of output channels (1 - 4)         -"-   1
   VALUE - INTEGER*4 - value to be assigned to the named parameter.
       There is a further restriction to this value, imposed by the
       maximum possible number of samples per frame:
         (sampling rate * frame duration * channels) / 1000
       may not exceed 2000
   STATUS - INTEGER*4 - gets:
                  1 OK
                 -1 unrecognized parameter name
                 -2 illegal value
                 -3 value OK, except that resulting frame size is too big
```

Error conditions: Message is written on unit 6, and no new values are assigned.
External call: MESS
For example of calling procedure, see TAPE.
!

3 DAGNW
!

level -1

Converts device parameter information to digital wave-form data and writes it on the disk file opened with DAGNI.

```
       CALL DAGNW
      - (SAMPNR, GCON, GFLAGS, GFREQ, GLEVEL, GPHASE, ALEVEL, STATUS)
```

SAMPNR - INTEGER*4 - the number of the sample at which ramps to the parameter values in this call are to end. This should normally be the number of the final sample in a frame; DAGNW calculates sound only when SAMPNR indicates that at least one whole frame is to be created.

GCON (256) - INTEGER*4 array - generator connection data

```
              0:  generator disconnected
        1 - 256:  connected to generator with this number
      257 - 512:  connected to CD number (GCON-256)
```

GFLAGS (256) - LOGICAL array - generator "active" flags

.TRUE. = active   .FALSE. = inactive

GFREQ (256) - REAL*4 array - generator frequencies

GLEVEL (256) - REAL*4 array - generator amplitudes in the range 0.0 to 1.0

GPHASE (256) - REAL*4 array - generator phases at the beginning of the frame, in the range 0.0 to 6.283185308. The values for active generators are updated automatically by DAGNW.

ALEVEL (256,4) - REAL array - amplifier amplitudes in the range 0.0 to 1.0

STATUS - INTEGER*4 - gets:

1 OK, calculations for at least one frame done, though there may be samples that have not yet been dealt with

2 OK, nothing done this time, but there are samples that haven't yet been dealt with

3 OK, nothing done, and nothing left to do (SAMPNR = number of last sample created)

-1 illegal sample number (must be => sample number in previous call to DAGNW)

-3 internal error in wave-form calculation (FGCH) - program package not initialized correctly

-4 either writing out of disk file range, or attempting to write to a file that has not been opened for writing (error in DASMW)

-6 OK, but excessive amplitudes (< -1.0 or > +1.0) have been found (in DASMW); a file containing the current device parameter values is written on unit 80 - see also under "Sound distortion"

External calls: DASMW, FGCH, MESS

For example, assuming a frame size of 500 samples:

```
  CALL DAGNW (100, GCON, GFLAGS, GFREQ, GLEVEL, GPHASE, ALEVEL, STATUS)
  CALL DAGNW (520, GCON, GFLAGS, GFREQ, GLEVEL, GPHASE, ALEVEL, STATUS)
  CALL DAGNW (1500, GCON, GFLAGS, GFREQ, GLEVEL, GPHASE, ALEVEL, STATUS)
```

The first call does nothing at all, since it specifies part of a frame only. The second call creates one frame up to sample #500, with ramps on generator parameters starting at zero and rising to the values specified in GFREQ, GLEVEL, etc. The third call creates two more frames, the first with ramps to the new generator parameters (samples #501 to #1000), and the second with static values (samples #1001 to #1500).

3 DARCE
!

level -3

Subroutine that closes a disk file, previously opened with DARCI

```
        CALL DARCE (FILENR, STATUS)

   FILENR - INTEGER*4 - file number (1 - 3)
   STATUS - INTEGER*4 - gets:
                 1 OK
                -1 illegal file number
                -2 file not open on this unit
                -3 I/O error on CLOSE
```

File number refers to the EMSDAC file-handling system. This is translated internally to logical unit number (80 + file number).

Error conditions: message is written on unit 6, & file is not closed.
External call: MESS

See DASME for example of calling procedure
!

3 DARCI
!

level -3

Opens a named file for reading or writing.

```
          CALL DARCI (FILENR, NAME, STATUS)

   FILENR - INTEGER*4 - file number (1 - 3)
   NAME - CHARACTER - name of file (including extension)
   STATUS - INTEGER*4 - gets:
                  2 OK, existing file opened
                  1 OK, new file opened
                 -1 illegal file number
                 -2 file with this file-number already open
                 -3 I/O error (on INQUIRE)
                 -4 I/O error (on OPEN)
```

Error conditions: message is written on unit 6, and file is not opened
External call: MESS

See DASMI for example of calling procedure
!

3 DARCR

!

level -3

Reads one logical record from a specified position in a disk file

```
        CALL DARCR (FILENR, RECNR, BUFSIZ, BUFFER, STATUS)

   FILENR - INTEGER*4 - file number (1 - 3)
   RECNR - INTEGER*4 - record number within the file
   BUFSIZ - INTEGER*4 - size of array BUF (must = 4096)
   BUFFER - INTEGER*2 array - receives data from the file
   STATUS - INTEGER*4 - gets:
                 1 OK
                -1 illegal file number
                -2 no file open on this unit
                -3 illegal BUFSIZ
                -4 I/O error on READ (maybe illegal RECNR)
```

Error conditions: If errors -1 to -3 are reported, the calling program's buffer remains unaltered. In the case of error -4, the previous contents of the buffer may be wholly or partly destroyed.
External call: MESS
See DASMW for example of calling procedure
!

3 DARCW

level -3

Writes one logical record to a specified position in a disk file.

```
        CALL DARCW (FILENR, RECNR, BUFSIZ, BUFFER, STATUS)

   FILENR - INTEGER*4 - file number (1 - 3)
   RECNR - INTEGER*4 - number of the record within the file to which
                 this data is to be transferred
   BUFSIZ - INTEGER*4 - size of array BUFFER (must = 4096)
   BUFFER - INTEGER*2 array - contains the data to be written to disk
   STATUS - integer variable - gets:
                  1 OK
                 -1 illegal file number
                 -2 no file open on this unit
                 -3 illegal BUFSIZ
                 -4 I/O error on WRITE
```

Error conditions: Message is written on unit 6. When errors -1 to -3 are found, control is returned immediately to the calling program: nothing is written to disk. With error -4, some or all of the record may have been written.

External call: MESS
See DASMW for example of calling procedure

3 DASME

!

level -2

Closes a disk file, after making the final transfer of any part-records left in the internal buffer.

```
        CALL DASME (NAME, STATUS)

    NAME - CHARACTER - name of the file to be closed (no extension)
    STATUS - INTEGER*4 - gets:
                  1 OK
                 -1 unrecognized file name
                 -2 file not open
                 -3 I/O error (in DARCW)
                 -4 I/O error (in DARCE)
```

Error conditions: file is not closed when negative codes are returned
External calls: DARCE, DARCW, MESS
For example of calling procedure, see DAGNE
!

3 DASMI
!

level -2

Opens a disk file for reading or writing. Allows three files to be open at the same time.

```
        CALL DASMI (NAME, DIREC, STATUS)

   NAME - CHARACTER - file name (without extension)
   DIREC - CHARACTER - I/O direction:
                 'READ' = access or update an existing file
                 'WRITE' = create a new file
   STATUS - INTEGER*4 - gets:
                 1 OK
                 2 OK, existing file opened
                 3 OK, file already open
                -1 no room for this file: 3 files already in use
                -2 illegal I/O direction (not 'READ' or 'WRITE')
                -3 error in DADAT (should never occur)
                -4 error in call to DARCI (I/O ERROR)
```

Error conditions:
Message is written on unit 6, and file is not opened.

External calls: DADAT, DARCI, MESS

See DAGNI for example of calling procedure
!

3 DASMR

!

level -2

Subroutine that fetches sampling data from a disk file, and converts it to floating-point format in the range -1 to +1. Any amount of data can be fetched from any part of a file with one call.

This routine is not in fact used in the EMSDAC file-writing package; it is included here for the sake of completeness. For examples of its use, see the documentation file EMSTREAT.HLP.

```
        CALL DASMR (NAME, SCOUNT, SPOS1, BUFFER, STATUS)
```

```
  NAME - CHARACTER - file name (max. 6 characters, no extension)
  SCOUNT - INTEGER*4 - number of data items to be fetched; must be
                zero or positive, and must = (number of
                channels * number of samples)
  SPOS1 - INTEGER*4 - position in file of first data item to be fetched
                - must be >0.  The first element in the file has
                number 1; thereafter, element number =
                  (sample number - 1) * number of channels + 1
  BUFFER - REAL*4 array - receives floating-point data between -1 and +1
  STATUS - INTEGER*4 - gets:
                 1 OK
                -1 unrecognized file name
                -2 file not open
                -3 illegal SCOUNT or SPOS1
                -4 I/O error (in DARCR or DARCW)
                -6 OK, but excessive amplitude(s) found in file
```

External calls: DABL, DARCR, DARCW

!

3 DASMW

!

level -2

Transfers floating-point sampling data in the range -1 to +1 to a specific place in a disk file, defined by sample number. Internally the data is converted to 16-bit format with a call to BLDA, and then gathered into records of 4096 INTEGER*2 words before being sent on to level -3.

EMSDAC always writes to files sequentially, with each call to DASMW writing one frame of sound to the file. DASMW itself, however, has no such restrictions: any amount of data can be written to any part of a file at any time.

```
        CALL DASMW (NAME, SCOUNT, SPOS1, BUFFER, STATUS)

   NAME - CHARACTER - file name (no extension)
   SCOUNT - INTEGER*4 - the number of data items to be written to disk;
                must be zero or positive, and must = (number of
                channels * number of samples)
   SPOS1 - INTEGER*4 - position within the file (i.e. element number) to
                which the first data item is to be written.  Must be >0.
                The first element in the file has number 1; thereafter,
                element number =
                  (sample number - 1) * number of channels + 1
   BUFFER - REAL array - calling program's array containing wave-form
                data in the range -1.0 to +1.0
   STATUS - INTEGER*4 - gets:
                 1 OK
                -1 unrecognized file name
                -2 file not open
                -3 illegal SCOUNT or SPOS1
                -4 I/O error DARCW
                -6 I/O performed OK, but excessive amplitudes found in
                   BUFFER (i.e. less than -1 or greater than +1)
```

Error conditions:
If error codes -1 to -3 are returned, no sampling information is converted or written to disk. When -4 is returned, some samples may have been written to disk, but it will not be possible to write more information; an attempt should be made to close the file with DASME. Code -6 is not a fatal error; calculations continue, though the resulting sound will be distorted.

External calls: BLDA, DARCW, DARCR, MAX, MESS, MIN

See DAGNW for example of calling procedure

!

3 FGCH
!

level -1

Called by programs at level -1 to calculate one frame of digital sound from generator and amplifier parameters. Data is input as values for frequencies and levels at the beginning and end of one frame, together with certain other parameters: numbers of samples, generators, channels, etc.

```
      CALL FGCH (TYPE, NIPAR, IPARAM, NSTART, START, FINAL,
   -       NPHASE, PHASE, BUFSIZ, BUF, NDATA, STATUS)
```

TYPE - INTEGER*4 - defines the generator type; type 1 only exists at present: 256 interconnectable sine-wave generators and a maximum of 4 channel output amplifiers. The function of the remaining arguments is here described for type 1 generation - with other types, the arguments could have completely different meanings.

NIPAR - INTEGER*4 - number of elements in array IPARAM

IPARAM - INTEGER*4 array - contains:
(1) number of samples in frame
(2) number of generators
(3) number of channels
(4) number of channel distributors
(5...IPARAM(2)+4) generator connection data: each element points to either a channel distributor (IPARAM(2)+n), or another generator's input.

NSTART - INTEGER*4 - number of elements in START (and FINAL)

START - REAL*4 array - contains:
sampling rate, FG1 start freq [, FG2 start freq,...], FG1 start amplitude [, FG2 start amplitude,...], CHAN 1 start amplitude[, CHAN 2 start amplitude, ...]
at the beginning of the frame.
Organization of channel distributor amplitudes:
CHAN1 (CD1[, CD2, ...CDn]) [, CHAN2 (CD1[, ... CDn]), ...CHANn (CD1[, ...CDn])]

FINAL - REAL*4 array - contains parameters as in START, but with final values for sampling rate, and numbers of generators, channels, and channel distributors

```
*************************************************************
*** N.B. FOR MODULATING GENERATORS, ARRAYS 'START' &      ***
*** 'FINAL' CONTAIN FREQUENCY DEVIATION IN HZ INSTEAD OF ***
*** AMPLITUDE                                             ***
*************************************************************
```

NPHASE - INTEGER*4 - number of elements in array PHASE

PHASE - REAL*4 array - generators' phase at the beginning of this frame, updated automatically on return

BUFSIZ - INTEGER*4 - maximum size of array BUF

BUF - REAL*4 array - receives floating-point sampling data

NDATA - INTEGER*4 - receives value describing number of data elements transferred to BUF; should = COUNT(1)*COUNT(3)

STATUS - INTEGER*4 - gets:
 1 OK
-1 illegal TYPE (must = 1)
-2 illegal NIPAR (must = 4+GCOUNT in TYPE 1)
-3 illegal number of samples
-4 illegal number of generators
-5 illegal number of channels
-6 illegal number of channel distributors
-7 illegal NSTART: must = 1+(GCOUNT*2)+(NCHANS*NCDS)

```
                 -8 illegal sampling rate
                 -9 illegal NPHASE (must = number of generators)
                -10 caller's buffer too small (BUFSIZ)
Error conditions:
All errors cause the program to return without writing new
values to BUF or updating PHASE.  Message is written on unit 6.

External calls: SIN, MOD, MESS

With patience and care, it should not be too difficult to construct new
generator types inside or outside FGCH, using the same arguments:

 * TYPE - different numbers for each generator type
 * INPUT arguments:
     1. control data in an integer array IPARAM, of length NIPAR
     2. initial and final parameter data in two real arrays
        START and FINAL, of length NSTART
 * UPDATED argument:
     parameter information in a real array PHASE, of length NPHASE
 * OUTPUT arguments:
     1. a real array BUF, of length BUFSIZ
     2. two integer variables, NDATA and STATUS, for control data

Note that FGCH has internal arrays that limit the total number of
generators and amplifiers in the system.  At present, the limit is set
at 256 generators, 256 channel distributors, and four amplifiers.

See DAGNW for example of calling procedure
!
```

3 update

!

This section contains hints to programmers on how to add new routines to the EMSDAC program package. We take an actual example of a routine that might be useful, and show what steps must be taken.

Suppose that we need a subroutine that sets amplitudes on individual generators. We wish to define the amplitudes as floating-point numbers on a linear scale between 0.0 and 1.0, and we want to be able to use the same routine for setting modulation indexes in the range 0.0 to 100.0; we do not wish to have to set frequencies and wave-forms at the same time. We'll give it the general appearance FGA (NR, AMP), where NR is the generator number and AMP is the amplitude.

The following tasks must be carried out:

1 RESEARCH
EMSDAC documentation and source files must be studied to answer the following questions:
- which program level is to be operated on? (e.g. generator phase and frequency are stored at level 0; file structure is dealt with at level -3, and so on)
- which COMMON BLOCKS hold the data that we wish to manipulate? the contents of common blocks are described in the source files for BLOCK DATA programs at each level
- how is the data formatted?
- what conventions do the other programs at this level follow in their treatment of the data? how are errors dealt with?

2 DOCUMENTATION
The new routine must be documented in the documentation file EMSDAC.HLP, and the help library file HELPLIB.HLB must be updated. See the VAX/VMS Utilities Reference Manual for information on updating library files.

3 CODING & COMPILATION
The routine is coded (preferably in Fortran), compiled, and the object module added to the library file DACLIB.OLB; all modules except those at level -1 should also be added to APDACLIB.OLB if the program package EMSAP is to function in the same way as EMSDAC. There is more information on this in EMSAP.HLP.

For our example program, FGA, we discover that generator amplitudes are stored at level 0 as floating-point numbers in common block /GENTAB/. We also discover that all routines at level 0 that store generator parameter data have certain things in common:

- they are all declared as INTEGER FUNCTION subprograms that return a code describing the success of the operation
- they check that device numbers and parameter values are legal, and take appropriate action if they are not
- they allow the use of the no-parameter flag (NOP) to indicate that no change is to be made to the parameter in question
- they set generator-active flags in common block /FLGTAB/ to show which generators are actually being used
- they update NGENS in common block /GENTAB/ - this stores the number of the highest numbered generator used since TAPE was called; this information is not used at present, though it is envisaged that it may be needed at some time in the future to speed up wave-form calculations.

!

Here, then is the program code that follows all the above conventions:

```
      INTEGER FUNCTION FGA (NR, AMP)

* TYPE DECLARATIONS: WE ALWAYS ASSUME THAT SYMBOLIC NAMES ARE
*             OF TYPE INTEGER*4 UNLESS THEY ARE DECLARED OTHERWISE.
*             'TEXT' IS USED TO STORE ERROR MESSAGE STRINGS.
*             THE OTHER NAMES DECLARED HERE ARE ARGUMENTS,
*             PARAMETERS, OR COMMON BLOCK ARRAYS.
* PARAMETERS: NFGS - MAXIMUM LEGAL GENERATOR NUMBER
*             MAXAMP - MAXIMUM LEGAL AMPLITUDE.  THIS WOULD NORMALLY
*               BE IN THE RANGE 0.0 TO 1.0, BUT WE PERMIT UP TO 100.0
*               TO ALLOW FOR MODULATION INDEX; THE USER MUST TAKE
*               RESPONSIBILITY FOR AMPLITUDES GREATER THAN 1.0
*             NRERR & AMPERR - ERROR CODES TO BE RETURNED AS FUNCTION
*               VALUES IN CASE OF ILLEGAL GENERATOR NUMBER OF AMPL.
*               WE FOLLOW THE CONVENTIONS OF 'EMSDEV'.
*             PRGLEV - PROGRAM LEVEL, NEEDED FOR CALL TO 'MESS'
*             ANOP - NO-PARAMETER FLAG
* COMMON BLOCKS: /GENTAB/ CONTAINS GENERATOR PARAMETERS (FREQ & AMPL)
*                /FLGTAB/ CONTAINS "ACTIVE" FLAGS

      IMPLICIT INTEGER (A - Z)
      REAL AMP, GFREQ, GLEVEL, MAXAMP, ANOP
      CHARACTER TEXT*40
      LOGICAL GFLAGS

      PARAMETER (NFGS=256, MAXAMP=100.)
      PARAMETER (NRERR=-21, AMPERR=-23)
      PARAMETER (PRGLEV=0, ANOP=-131070.)

      COMMON /GENTAB/ NGENS, GFREQ (NFGS), GLEVEL (NFGS)
      COMMON /FLGTAB/ GFLAGS (NFGS)

* PROGRAM FLOW
* ------------
* IF ILLEGAL GENERATOR NO., WRITE MESSAGE, SET FUNCTION VALUE & RETURN
* IF AMP = -131070.0 THEN OKAY, BUT DO NOTHING EXCEPT RETURN (WITH
*   FUNCTION VALUE 2)
* NOW WE KNOW THAT WE'RE GOING TO CHANGE AMPLITUDE, SO MARK GENERATOR
*   AS "ACTIVE" IN GFLAGS, AND UPDATE 'NGENS'
* IF AMP OUTSIDE LEGAL RANGE, ADJUST TO NEAREST LIMIT (0 OR MAXAMP), &
*   PUT IN 'GENTAB'; WRITE MESSAGE, & SET FUNCTION VALUE TO 'AMPERR'
* IF BOTH NR & AMP ARE WITHIN LEGAL LIMITS, SET FUNCTION VALUE TO 1,
*   PUT AMP IN 'GENTAB', & RETURN.
```

```
         IF ((NR .LE. 0) .OR. (NR .GT. NFGS)) THEN
            STATUS = NRERR
            WRITE (TEXT, 991) NR, ': ILLEGAL GENERATOR NUMBER'
            CALL MESS (STATUS, 'FGA', PRGLEV, TEXT)

         ELSE IF (AMP .EQ. ANOP) THEN
            STATUS = 2

         ELSE
            GFLAGS (NR) = .TRUE.
            NGENS = MAX (NGENS, NR)

            IF ((AMP .LT. 0.) .OR. (AMP .GT. MAXAMP)) THEN
               STATUS = AMPERR
               WRITE (TEXT, 992) AMP, ': ILLEGAL AMPLITUDE'
               CALL MESS (STATUS, 'FGA', PRGLEV, TEXT)
               IF (AMP .GT. MAXAMP) THEN
                  GLEVEL (NR) = MAXAMP
               ELSE
                  GLEVEL (NR) = 0.
               ENDIF
            ELSE
               GLEVEL (NR) = AMP
               STATUS = 1
            ENDIF
         ENDIF

* SET FUNCTION VALUE AND RETURN

         FGA = STATUS
         RETURN

* FORMAT STATEMENTS FOR ERROR MESSAGES

991   FORMAT (I12, A)
992   FORMAT (F12.3, A)
      END
```