MAX LANGUAGE REFERENCE MANUAL

June 15, 1982

The material in this manual is for informational purposes only and is subject to change without notice.

New England Digital Corporation assumes no responsibility for any errors which may appear in this manual.

CONTENTS

USING THIS MANUAL

The MAX Reference Manual is a compact, but complete description of the MAX language. Experienced programmers may turn to this Manual immediately to learn the details of all MAX procedures.

All MAX procedures are summarized as well as used in a format statement which demonstrates the use of the procedure. In these statements we have adopted certain simple conventions. Examine the following:

```
ln chan = ALLOCATE;
```

Upper case is used to indicate XPL/4 or MAX reserved words (ALLOCATE). Lower case is used to indicate variables or numbers supplied by the user (ln, chan). The meaning of user variables is defined in the summary paragraph above each format statement. The expression ln always means linenumber.

Note also that in every case except one the values either passed or returned in MAX procedures are FIXED point values. The single FLOATING point exception is the octave-point pitch-class argument accepted in the PCH frequency conversion function (see page 11).

SYSTEM OPERATION

XPL/4 MONITOR

The standard XPL/4 monitor is used to manage MAX files and to compile and run MAX programs. The XPL/4 monitor is basically the same as the SCRIPT monitor, except that where the SCRIPT monitor CONVERTS and PLAYS SCRIPT compositions, the XPL/4 monitor COMPILES and RUNS XPL or MAX programs. The file management commands are identical with either monitor. See the SCRIPT Reference Manual for a summary of these commands.

XPL/4 Performance Commands

RUN — The RUN command compiles and initiates execution of a MAX program or composition.

COMPILE — The COMPILE command translates the MAX program into a form which is immediately ready for execution. The compiled form of the program replaces the current file; the current file name is changed by the monitor by adding a period (.) to the end of the current file name. The compiled program may be saved on the user diskette by use of the SAVE command. The RUN command may be used to execute a compiled program.

MAX USER DISKETTES

MAX user diskettes contain the three sections of the MAX library in two forms: the compacted version (MAXSYN, MAXIO, MAXTASK) and the source version (MAXSYN1, MAXIO1, MAXTASK1). The compacted version has had all comments and line numbers removed for faster compilation. The source version has many comments and can be listed and modified by the user.

MAX user diskettes also contain fifteen demonstration programs, as well as system file .STAB-4. All these files are stored on a single MAX user diskette with one exception. For users of single density 5 1/4 diskettes, the files have been stored on four separate diskettes as follows:

a. User Diskette #1 - MAXSYN, MAXIO, and MAXTASK and Demo1-Demo10
b. User Diskette #2 - MAXSYN, MAXIO, and MAXTASK and Demo11-Demo14
c. User Diskette #3 - MAXSYN, MAXIO, and MAXTASK and Demo15
d. Source Diskette - MAXSYN1, MAXIO1, and MAXTASK1

Creating New Max User Diskettes

You can use the XPL/4 FORMCOPY utility program to format a new blank diskette and to duplicate a MAX user diskette onto it. Instructions may be found in the Scientific XPL/4 Documentation Update.

You will probably wish to UNSAVE the demonstration examples as well as the MAX source files on one copy of your MAX user diskette before using it as a master diskette.

If you use minidiskettes, you may also wish to UNSAVE one or all of the compacted MAX files on certain of your user diskettes to free up extra space for your own compositions. In this case, when you run your MAX composition, the monitor will ask you to insert a diskette which contains the file(s) specified in the INSERT statement(s) into the right-hand drive during compilation. Swapping diskettes in this way is time-consuming, but may save crucial space on a minidiskette.

If you have a maxidiskette system, you may wish to store both SCRIPT and MAX files, as well as other programs, on one diskette. There is no problem in doing this (as long as you have enough space). To compile and play SCRIPT user files, the "hidden" SCRIPT system files, such as .BNKDATA and .SCON-4, must be present.

STRUCTURE OF THE MAX LIBRARY

There are three sections to the MAX library.

MAXSYN — This section contains the procedures used for controlling the digital synthesizers, such as frequency control, waveform memory loading and selection, frequency modulation, envelope specification, etc.

MAXIO — This section contains the input and output procedures for use with the Synclavier (R) II keyboard control unit and the pedals.

MAXTASK — This section contains the procedures for creation of parallel tasks and processes.

The sections are included in a MAX program by way of an INSERT statement in this form:

```
ln INSERT 'filename';
```

MAXSYN PROCEDURES

MAXSYN procedures provide the user with the means to send control code to the channels in the digital synthesizers.

ALLOCATING A CHANNEL

The term channel is synonymous with voice. The channels are numbered by even numbers starting with 0. For example, the channels for an 8-voice system are numbered 0, 2, 4, 6, 8, 10, 12, and 14.

The following procedures are related to channel allocation and setup.

ALLOCATE — Selects a channel from list of free channels. Is passed no arguments. Returns a channel number (chan). The MAXSYN initialization code creates an initial list of free channels which equals the number of voices in your system.

```
ln chan = ALLOCATE;
```

ALLOCATEX — A special form of ALLOCATE. Is passed an array of channel numbers (channellist). Returns a channel number from this list. The zeroeth element in the array is the number of channel numbers in the list. The other elements are the channel numbers. This function allows the output from specific channels to be directed to particular output jacks in stereo or quad systems.

```
ln chan = ALLOCATEX(channellist);
```

FREECHAN — Returns a channel number to the list of free channels. Is passed a channel number. Is called after usage of a channel is completed.

```
ln CALL FREECHAN(chan);
```

CLEANUP — Sets all parameters for a channel at zero, thus silencing it. Is passed a channel number. Also resets the wave memory pointer for the channel to select the sine wave memory.

```
ln CALL CLEANUP(chan);
```

ZEROSYN

Performs a CLEANUP on all channels. Is passed no arguments. Is called during MAXSYN initialization code to put the synthesizer in a known state.

ln CALL ZEROSYN;

FREQUENCY SPECIFICATION

Setting the Frequency

There are two procedures which can be called to set the frequency for a channel. In the first the frequency of the carrier is directly specified and the frequency of the modulator is specified by way of an FM ratio. In the second both carrier and modulator frequency are directly specified. Procedures for precalculation and fast emitting are also available.

SETFRQ

Calculates 24-bit frequency descriptor and emits it to the synthesizer channel. Is passed channel number (chan), frequency number (freq.num), and ratio. The frequency number is a value returned from any of the four conversion functions described below. The ratio is an integer 1000 times the desired FM ratio.

ln CALL SETFRQ(chan,freq.num,ratio x 1000);

SETFRQ2

Calculates 24-bit frequency descriptor and emits it to the synthesizer channel. Is passed channel number (chan), frequency number for the carrier (carfnum), and frequency number for the modulator (modfnum). The frequency numbers are values returned from any of the four functions.

ln CALL SETFRQ2(chan,carfnum,modfnum);

CALCFRQ

Calculates 24-bit frequency descriptor and stores it in global variables NOTEADD, NOTEDIV, and NOTEINC. Is passed frequency number from one of the four conversion functions.

ln CALL CALCFRQ(freq.num);

EMITEFRQ

Emits carrier frequency to synthesizer channel. Is passed channel number and NOTEADD, NOTEDIV, and NOTEINC.

```
ln CALL EMITEFRQ(chan,NOTEADD,NOTEDIV,NOTEINC);
```

EMITIFRQ

Emits modulator frequency to synthesizer channel. Is passed channel number and NOTEADD, NOTEDIV, and NOTEINC.

```
ln CALL EMITIFRQ(chan,NOTEADD,NOTEDIV,NOTEINC);
```

Frequency Conversion Functions

There are four alternative functions for converting frequency or pitch into the internal frequency code which can be used in the SETFRQ procedures. The first two specify absolute frequencies. The third and fourth are affected by calls to SET.TUNING.BASE and by the value stored in the variable OCTAVE.RATIO.

HERTZ

Returns frequency number code (freq.num). Is passed an integer ten times the frequency in hertz.

```
ln freq.num = HERTZ(frequency x 10);
```

This function is absolute.

PCH

Returns frequency number code. Is passed a FLOATING POINT octave-point pitch-class number. The number to the left of the decimal indicates the octave, the first two digits to the right of the decimal indicate the pitch in semitones above C and any following digits indicate tenths, hundredths, etc., of semitones above C.

```
ln freq.num = PCH(octave-point pitch-class number);
```

For example, PCH(8.00) returns a frequency number code equivalent to middle C and PCH(8.04) returns a frequency number equivalent to the E above middle C.

This function is absolute. It is also the ONLY procedure in MAX which is passed a FLOATING point value.

PITCH

Returns frequency number code. Is passed a character string in SCRIPT notation. Each pitch is indicated by a pitch letter followed by an optional sharp symbol (#) or flat symbol (F) and octave number (1 through 5). Accidental symbols do not affect subsequent notes. Octave numbers apply to succeeding notes until a new octave number is entered after the pitch. The default octave is 3. Double and multiple accidentals are allowed.

ln freq.num = PITCH('SCRIPT pitch');

This function i not absolute. It is affected by the tuning base and octave ratio values.

KEY

Returns frequency number code. Is passed an integer indicating a key number on the Synclavier (R) II keyboard. The lowest key (lowest C) on the keyboard is numbered 0. The highest is 60. Middle C is 24.

ln freq.num = KEY(keynumber);

This function is not absolute.
It is affected by the tuning base and octave ratio values.

Special Tuning

SET.TUNING.BASE

Sets the tuning base. Is passed an integer ten times the frequency in hertz.

ln CALL SET.TUNING.BASE(frequency x 10);

It affects all PITCH and KEY frequency conversions.

OCTAVE.RATIO

A variable in which is stored an integer 1000 times the octave ratio. The default OCTAVE.RATIO value is 1000.

ln OCTAVE.RATIO = octave ratio x 1000;

This setting affects all PITCH and KEY frequency conversions.

SETTING THE VOLUME

SETVOL

Places 8-bit value in volume register of synthesizer channel. Is passed a channel number and a value between 0 for no volume to 255 for maximum volume.

ln SETVOL(chan,value);

SETISHC

Emits shift count to index interpolator in synthesizer channel. Is passed channel number and shift count value in the range between 0 and 3. The index shift count changes the index limit as follows:
With an index shift count of 0, the index limit will be as specified in the SETILIM, EMITILIM, or SETI statements. With a shift count of 1, the index limit will be multiplied by 2. With a shift count of 3, the index limit will be multiplied by 4 and with a shift count of 4, the index limit will be multiplied by 8.

ln CALL SETISHC(chan,shiftcountvalue);

EMITISHC

Emits shift count to index interpolator. Is passed channel number and shift count value as described above. Exactly the same as SETISHC.

ln CALL EMITISHC(chan,shiftcountvalue);

WAVEFORM MEMORIES

The synthesizer has 32 waveform memories which are shared between the channels. One of these memories is preset to contain a sine wave. The wave memory pointers for all channels are initialized to select this memory. MAXSYN provides procedures for calculating any complex waveform, emitting it to one of the waveform memories, and linking a specific waveform memory with a specific channel.

In the procedures below, the expression harmonic coefficients array (harmcoefarray) refers to an array containing integers in the range from 0 to 1000 indicating the relative amplitudes of the harmonics. (These numbers correspond to the DIGITAL∿TONE GENERATORS settings of 0.0 to 100.0 in the Synclavier (R) II real-time system.) The zeroeth element in the array indicates the number of coefficients specified.

SETWAVE

First checks to see if any waveform memory contains desired complex waveform. If so, returns that waveform memory number (wavemem#). If not, calculates complex waveform, emits waveform array to waveform memory, and returns that waveform memory number. Is passed harmonic coefficients array. Also counts the number of current uses of a waveform memory.

ln wavemem# = SETWAVE(harmcoefarray);

CALCWAVE

Calculates a complex waveform and stores it in the 256-point global array WAVEBUF. Is passed a harmonic coefficients array.

```
In CALL CALCWAVE(harmcoefarray);
```

EMITWAVE

Emits 256-point waveform array to waveform memory in synthesizer. Is passed wave memory number (wavemem#) and waveform array (wavearray).

```
In CALL EMITWAVE(wavemem#,wavearray);
```

FREEWAVE

Frees up a usage reference for a wave memory. Is passed a waveform memory number.

```
In CALL FREEWAVE(wavemem#);
```

SETWSEL

Links channel with wave memory. Is passed channel number and waveform memory number.

```
In CALL SETWSEL(chan,wavemem#);
```

EMITWSEL

Links channel with wave memory. Is passed channel number and waveform memory number. Exactly the same as SETWSEL.

UTILITIES

The following utility procedures are also included in MAXSYN.

RND

Generates random numbers (rand#). Is passed two integers indicating a minimum/maximum range.
Returns an integer from the minimum to one less than the maximum.

```
In rand# = RND(min,max);
```

The RND function will always produce the same random number sequence for each run.

WAIT

Initiates an idle state for the processor for a specified period. Is passed a time period from 0 to 32,767 milliseconds.

```
In CALL WAIT(time);
```

LOCATE.FILE

Can be used to search for any file. Is passed a 256-point array (catbuf) read from sector 0 of diskette and a filename. Returns the starting sector number (sector) for the named file, if found.

```
ln sector = LOCATE.FILE(catbuf,filename);
```

This function is used in the MAXSYN initialization code to look for data file .STAB-4 on the system or user diskette.

MAXIO PROCEDURES

USING SCANDATA

The major MAXIO procedure, SCANDATA, controls the basic communication between the Synclavier (R) II Digital Synthesizer and the Synclavier (R) II keyboard unit and pedals. It uses the MAX variables described below to store input and output data and acts as the transfer agent between these variables and the outside world. There are eighteen input variables and ten output variables. Their names describe either their physical location or label on the Synclavier (R) II keyboard unit. It is important to note, however, that these variables have no intrinsic meaning. They do not function as they do in the Synclavier (R) II real-time system. The function of each variable is entirely determined by the MAX program.

The call to SCANDATA is usually placed within a loop which includes a WAIT, or a SUSPEND in multitask programs (see MAXTASK), of five milliseconds. This provides real-time synchronized response to user input while the program runs. The input variables are set by the call to SCANDATA. The information contained in them is accessed by the program after the call to SCANDATA. On the other hand, the output variables are set by the program before the call to SCANDATA.

The other MAXIO procedures offer convenient means to set and use the SCANDATA input and output variables.

INPUT VARIABLES

Each call to SCANDATA causes a scanning of the keyboard, buttons, control knob, ribbon controller if any and pedals if any. The results of this scanning operation are written in the variables below.

CLAVIER — An array with a 16-bit element for each octave of the keyboard. In each element the 12 lower-order bits indicate the status of the 12 keys in the octave. A zero bit indicates that the key is up, and a one bit indicates that the key is down.

PANSW — An array with a 16-bit element for each panel of 16 buttons on the Synclavier (R) II control panel. The zeroeth element in the array is the ENVELOPE panel of buttons, the seventh element is the RECORDER STORE/RECALL panel of buttons.

In each element, each bit corresponds to one of the buttons. The least significant bit is the upper left button in the panel. The most significant bit is the lower right button in the panel. A zero bit indicates that the button is not pressed. A one bit indicates the button is pressed. All pressed buttons are lit by SCANDATA.

KNOB.POS A variable indicating the current position of the control knob. When the knob is fully left, the value will be 100. When the knob is fully right, the value will be 160. (These numbers may be different by a few counts on some systems.)

KNOB.BASE A variable indicating the neutral, centered position of the knob, typically 130.

KNOB.CHANGE A variable that can be added to any parameter in order to change the parameter as the user turns the knob. It is the filtered and smoothed result of KNOB.POS-KNOB.BASE.

RTEPEDAL.POS A variable indicating the current position of the pedal connected to the jack labeled REAL TIME EFFECTS on the back of the Synclavier (R) II keyboard unit. A zero is written when the pedal is all the way up or turned off. A value of 225 is written when the pedal is all the way down.

VOLPEDAL.POS A variable indicating the current position of the pedal connected to the jack labeled OVERALL VOLUME on the back of the Synclavier (R) II keyboard unit. A zero is written when the pedal is all the way up or turned off. A value of 225 is written when the pedal is all the way down.

RIB.ACTIVE A variable indicating whether the ribbon controller is active (pressed) or not. The value is 1 when the ribbon is active, and 0 if not.

If active, the following two variables will also be set:

RIB.BASE A variable indicating the place on the ribbon that was first pressed. The range is from 15 for the left end to 200 for the right end. (As with the knob, these numbers may be different by a few counts on some systems.)

RIB.POS A variable indicating the place on the ribbon that is currently being pressed. The range is 15 to 200 as above.

PBI.POS
A variable determined by the voltage of the pitch bend input. The range is from 107 to 128 when a pedal is connected to the input, but may be a wider range for other devices.

PBI.BASE
A variable determined by the voltage of the pitch bend input at initialization time.
(For proper results, the pitch bend input device should be at its neutral position at initiation time.)

HOLD.SWITCH
REP.SWITCH
GLIDE.SWITCH
SUST.SWITCH
ARP.SWITCH
PUNCH.SWITCH
Variables indicating values for the input switches. They correspond to the HOLD, REPEAT, PORTAMENTO, SUSTAIN, ARPEGGIATE, and PUNCH IN/OUT input jacks on the back of the Synclavier (R) II keyboard unit. The value is 1 if the switch is connected and closed; otherwise, it is 0.

OUTPUT VARIABLES

The output variables are used by SCANDATA to represent data leaving the system. The user sets these variables before the call to SCANDATA.

DISPLAYSW
An array which indicates which buttons on the control panel are to be lit. The elements in the array are formatted in the same way as in PANSW. A 1 bit causes the button to be lit. A zero bit indicates the button will be turned off.

DIGDISPLAY
An array which indicates the elements of the LED which are to be lit. The user should call the special DISPLAY procedure described below to set up this array.

GATE.OUT
TRIGGER.OUT
CV.OUT
RIBBON.OUT
HPFILT.OUT
BPFILT.OUT
LPFILT.OUT
BANDWIDTH.OUT
Variables which produce control voltage outputs from the 8-bit DAC's on the back of the Synclavier (R) II keyboard unit. They correspond to the jacks labeled KEYBOARD GATE, KEYBOARD TRIGGER, KEYBOARD CV, RIBBON, HIGH PASS, BAND PASS, LOW PASS, and BANDWIDTH. The coding is such that a value of zero produces zero volts and a value of 255 produces 10.4 volts.

SPECIAL PROCEDURES

There are four procedures which aid in setting or using the variables associated with SCANDATA.

DISPLAY

Sets up the DIGDISPLAY variables so that the LED will display a value with a decimal point and one of the units lights to the right of the display will be lit. Is passed a value from 0 to 9999, an integer (decpos) indicating the number of digits to the right of the decimal point, and a units code (units). This units code is 1 for MILLISECONDS, 2 for HERTZ, 4 for ARBITRARY, and 8 for DECIBELS. More than one units light can be lit by adding the codes for each light desired. This procedure is called before the call to SCANDATA.

```
ln CALL DISPLAY(value,decpos,units);
```

DISPLAY.ERROR

A special case of DISPLAY used for error messages. Is passed a value between 0 and 9 and sets up DIGDISPLAY variables to show Err0 through Err9. This procedure is called before the call to SCANDATA.

```
ln CALL DISPLAY.ERROR(value);
```

SCAN.KEYBOARD

Returns the number (numkeys) of keys depressed since the last call to SCAN.KEYBOARD as well as their key numbers. Is passed an array (list) in which key numbers will be placed, starting with the zeroeth element. Is also passed a maximum size for the array (listsize). The size of the array determines how many new keys can be detected at one time. If no new keys have been depressed, the number returned (numkeys) will be zero. This procedure is called after the call to SCANDATA and uses the CLAVIER array.

```
ln numkeys = SCAN.KEYBOARD(list,listsize);
```

SCAN.RELEASE

Similar to SCAN.KEYBOARD. Returns the number (numkeys) of keys released since the last call to SCAN.RELEASE as well as their key numbers. Is passed and returns arguments exactly like SCAN.KEYBOARD. This procedure is called after SCANDATA.

```
ln numkeys = SCAN.RELEASE(list,listsize);
```

MAXTASK PROCEDURES AND STATEMENTS

MAXTASK allows the creation of parallel processes and multiple simutaneous events. Each event is defined as a task.

TASK DEFINITION

A task resembles a regular procedure with a few exceptions. The first line in the task takes this form:

```
ln taskname: PROCEDURE;
```

There can be no argument list associated with the task.

The last line in the task takes this form:

```
ln END taskname;
```

If there are to be several copies of a task active at once, local variables must be declared AUTOMATIC, to make them local to each individual task.

The AUTOMATIC type declaration takes this form:

```
ln DCL variablename AUTOMATICn;
```

where n is a number 1 through 6. Up to six variables may be declared AUTOMATIC within a task. Due to the nature in which AUTOMATIC variables are stored, they may not be used as indices of DO loops. Use a direct IF-THEN-GOTO loop.

THE MAIN TASK

There must be one primary task called MAIN. It is named just after the INSERT statements in the following way:

```
ln MAIN: PROCEDURE;
```

The last line in the program should be:

```
ln END MAIN;
```

STARTING, STOPPING AND SUSPENDING TASKS

A task is begun by means of a START statement in this form:

```
ln START taskname TASK;
```

The MAIN task is started by the MAXTASK initiálization code. All other tasks are started from within the MAIN task.

An optional SET clause may be inserted in the START statement to set values for the AUTOMATIC variables of the named task. This START statement takes this form:

```
ln START taskname SET(v1,v2,v3,v4,v5,v6) TASK;
```

where v1 through v6 are values to be stored in the six AUTOMATIC variables. All six values must appear even if there are not six AUTOMATIC variables.

A task may stop itself by a TERMINATE statement in this form:

```
ln TERMINATE TASK;
```

This is the normal ending of a task.

A task may stop another task by a KILL statement in this form:

```
ln KILL taskname TASK;
```

A task may suspend itself and go into an idle state for a specified length of time, thus giving other tasks a chance at the processor. The SUSPEND statement takes this form:

```
ln CALL SUSPEND(time);
```

Time is specified in milliseconds. Do not call SUSPEND with a time of less than five milliseconds (one clock tick). The SUSPEND procedure should be used instead of the WAIT procedure when you are using multiple tasks.

ERROR MESSAGES

The MAXTASK error messages are summarized here.

Error1: Too many tasks active. The default limit is 18. This limit may be expanded by increasing the value in the NUM.TASKS declaration in the MAXTASK source code.

Error2: Error in starting of task. The START statement has an incorrect format.

Error3: Stack length exceeded. Correct this problem by increasing the value in the LEN.STACK declaration in the MAXTASK source code. Refer to the Scientific XPL/4 Reference Manual for more information on push down stack requirements (PDL).

Error4: All tasks terminated. This error is generated when all tasks have been terminated. This may be a normal result in some instances.

Error5: Error in killing of task. The KILL statement has an incorrect format.

PROBLEM SOLVING

The XPL/4 compiler provides an excellent set of diagnostic messages, and will point out the location of an error. The compiler, however, is unaware of the special constructs and variables of MAX and cannot tailor messages especially for the MAX user.

The most common error is accidental usage of a variable identifier which has already been used in the MAX source code, for example CLAVIER or KEY. To help prevent the user from accidentally hitting on MAX words, all internal MAX variables have been given names containing periods and/or section prefixes, such as OLD.CLAVIER.K, MAXSYN.CLOCK.DIVISOR, or MT.TIME.

Do not use the words LOAD, MUL, ADD, or DIV, as these are part of the MAXSYN instructions for the hardware multiply/divide board.

For problems related to operation of the computer terminal, refer to the problem solving section in the SCRIPT documentation.

Remember that if you include the MAXIO section of the library, you must have a Synclavier (R) II keyboard unit connected to the system. If problems in execution occur when you have included MAXIO, first check the keyboard unit connections.