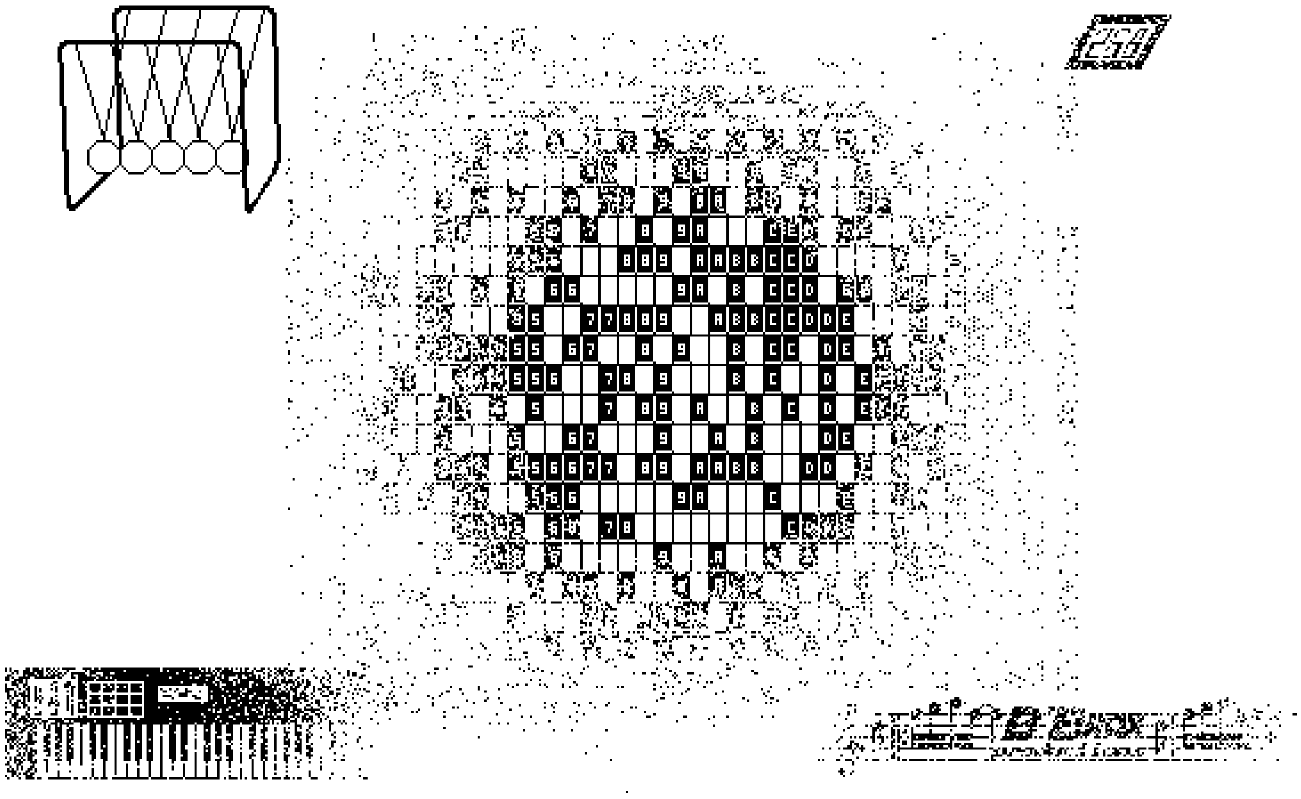
PATTERNER 1.3

Midi Music Experimental Kit

Manual



For ATARI ST monochrome and color systems

B'BROX
productions

THE CYCLE SYSTEM 90

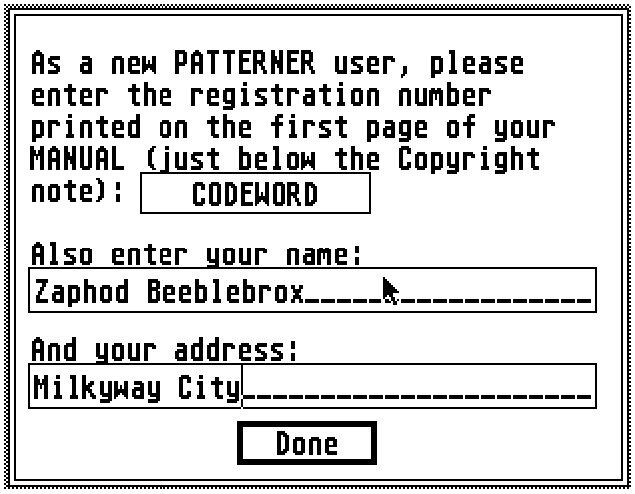
0. QUICK START

0.0. Backing up your program disk

Before doing anything else with your program disk (the one which comes with this book) you should make a backup copy. Since there is no copy protection on the disk you can use every disk copy program you like. However, simply copying the program file PATT_C.PRG or PATT_M.PRG won' t do. You have to copy the complete disk (using for example the disk copy program built in in the ST' s Desktop - you' ll find more on this in your ST' s manual).

0.1. Initializing the PATTERNER keydisk

If you try to run PATTERNER for the first time from your keydisk (a copy of it, I hope!) you will have to enter the 8 digit user registration number along with your name and address into this GEM box:


As a new PATTERNER user, please enter the registration number printed on the first page of your MANUAL (just below the Copyright note): CODEWORD

Also enter your name: Zaphod Beeblebrox

And your address: Milkyway City

You find your user registration number on the first page of this manual just below the copyright note. All eight digits have to be entered. Correct case is important. After you have entered all requested data press <*RETURN*> or click over *DONE*. If you entered a wrong registration number, the box pops up again and you should check all eight digits.

If you have entered the correct number your keydisk will be initialized and the PATTERNER **MAINPAGE** will show up in a second. Note that your registration number is also used in connection with your user registration card which entitles you to our update service. From now on this keydisk is initialized and starting PATTERNER from this disk will always get you straight to the **MAINPAGE**.

0.2. Use of the keydisk/Harddrive installation

Both PATT_C.PRG and PATT_M.PRG can be copied to harddrives and RAMdisks and also started from there by double-clicking. This, however, will bring up a small info on the screen urging you to insert the keydisk into drive A. After inserting the requested disk and clicking the *DONE* button the program will run properly. If your keydisk is not yet initialized at this point refer to chapter 0.1. Keep in mind that ANY copy of a keydisk, done with a disk copy is a valid keydisk.

0.3. Minimum Hardware Requirements

- ATARI 520/1040/MEGA or STACY Computer

- Single sided disk drive

- Monochrome- or Color monitor/TV

- MIDI keyboard capable of sending and receiving MIDI Note On and Note Off messages

- Two MIDI connection cables (DIN)

0.3.1. Optional Equipment

- More MIDI keyboards (MIDI Thru ports required).

- MIDI keyboards recognizing velocity data.

- MIDI keyboards recognizing program changes.

- Other MIDI equipment (signal processors, sound generators...).

0.4. Software Requirements

- main program file PAT_M.PRG (monochrome) or PAT_C.PRG (color)

- ***CYCLER*** assembler routine library, CYCLER.LIB

- ***CYCLER*** routine library source code, CYCLER.Q

0.5. Get Started

- Connect your MIDI keyboard' s MIDI In with the ST' s MIDI Out port

- Connect your MIDI keyboard' s MIDI Out with the ST' s MIDI In port

- Connect further keyboards to MIDI Thru of previous keyboard

- (don' t forget to switch everything on!)

- Insert the program disk into drive A

- Double-click PAT_M.PRG/PAT_C.PRG program file

- Now you are on the **MAINPAGE** of the PATTERNER

- Select *LOAD SYSTEM* in the *FILE* menu

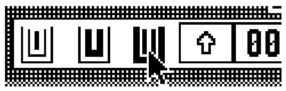
- Load one of the offered demo systems

- Exit the following *FILE NAME BOX* using the *CONTINUE* button

- Wait for the **SYSTEM** to be loaded

- Playback the loaded **SYSTEM** using the <*S*> key on the keyboard or **SONG** button in the **PLAYBACK** menu.

SONG Playback Button



- Change speed while playing back or try other functions (drawing in the **GRID WINDOW**...)

- Playback will stop if you:
a) Hit <*ESC*> on the ST keyboard
b) Left-click the mouse over the *STOP* button.
c) Click over the *PAUSE* button on the **s**

Playback is interrupted while LOADing or SAVing files. After SAVing playback continues. LOADing aborts playback completely.

1. BASIC CONCEPT OF THE PATTERNER

1.1. What can you do with it?

- You CANNOT play your MIDI keyboard and let the program record your performance - it‘s not a sequencer

- You CANNOT enter music in music notation nor print it

- You can use it as an advanced drum machine

- You can create endless rhythmic and melodic variations and playback the patterns using different tonal systems

- You can use up to sixteen synthesizers, each responding to one of the sixteenSOUND **CHANNELs**

- All MIDI data can also be routed to the ST' s internal sound chip

- You can also setup the PATTERNER to react to the music you play on the connected MIDI keyboard

- In addition you can control all MIDI gear which is able to receive continous data such as drum machines, MIDI expanders, effects processors...

- You can enter 16 character names for all structures and substructures you are dealing with

- You can enter ' conventional music' and change it in multiple ways. The PATTERNER' s presentation of data is different and forces you to think different

- You can alter all playback parameters and data while playback is running

- You can also setup PATTERNER to do the changes by itself

- You can use **GRAPHIC EDIT** tools to change music data

1.1.1 This is what it does...

The PATTERNER works with all kinds of MIDI data, mostly MIDI Note On and MIDI Note Off messages. When a MIDI keyboard receives a Note On message you will hear a sound. This sound is (normally) held until the keyboard receives the corresponding MIDI Note Off message.

On the other side: If you press a key on your MIDI keyboard it sends a MIDI Note On message followed (normally) by the respective MIDI Note Off when the key is released.

A MIDI Note On command contains the following information:

MIDI channel (0-15)
MIDI keynumber (number from 0-127 to define the pitch)
MIDI velocity (number from 0-127 to define the volume)

A MIDI Note Off command contains this:

MIDI channel
MIDI keynumber
MIDI velocity

(As usual computers and digital equipment always start counting from zero (0). Within PATTERNER all numbers are entered and displayed starting from zero. Always keep in mind that some MIDI devices display MIDI channels or other data starting with zero and others starting from one (1). You can find out about the display habits of your MIDI devices by dialing the lowest possible value of a parameter (is 0 or 1?). There are a few exceptions to the ' start from 0' -rule within PATTERNER.)

The basic core of the PATTERNER works with these three MIDI parameters. Unlike in a sequencer the messages are created, edited, stored and saved seperately in different **BANKS**. One **BANK** only contains data on when and how loud a Note On message will occur (**BEAT BANK**). Another one contains only MIDI keynumbers and MIDI channels which are assigned to the Note On messages at playback time (**SCALE BANK**). While the previous two **BANKS** are essential for playback the following one is used to send auxilary data (**MIDI Macro BANK**). This contains structures which let you send MIDI data of all different kinds to any of the connected MIDI devices at specific times during a performance.

The PATTERNER' s work consists mainly of putting together complete MIDI Note On/Off messages from the available data using many different patterns and rules.

The PATTERNER contains countless routines which let you influence the creation of MIDI messages, including a system of programmable subroutines which are controlled by playback itself.

1.2. GEM Objects and how they work

Here are some guidelines on how to work with the ST' s mouse, GEM boxes and objects in PATTERNER.

1.2.1. Mouse Pointer

Moving your mouse will move an arrow across the screen according to the directions and the speed you move the mouse. Usually nothing happens if you just move the mouse pointer without clicking one of the mouse buttons. However, if you move the mouse pointer into the upper region of the screen onto one of the menu titles, the respective menu folds down. You can move the pointer within the menu. Selecteable items appear in reverse if the pointer is over them. Click the left mouse button to activate the option you need. If you don' t want to make a selection in the menu just left-click with the pointer outside of the menu box, and it disappears again.

1.2.2. Mouse Clicking

Except in the menubar [1.2.1.] there is usually not much action unless you push one of the mouse buttons. There are several clicking patterns used in this program. The following descriptions show only a few examples. They are not intended as a complete list. Refer to the chapter numbers in brackets for detailed information.

a) Single left-click

Most functions are launched by placing the mouse pointer over an object (button, textfield, menu entry and so on) and then left-click once.

In the PATTERNER, left-clicking over the **BEAT EDIT GRID** [2.1.] also fills/clears **GRID POSITION**s.

You can also left-click over buttons to switch on/off functions (=> *PAUSE* button on **MAINPAGE** [3.7.]). Buttons which are switched on appear in reverse video.

Also, left-clicking over text fields like in the **BANK SELECTOR BOX**[4.2.2.], **PATTERN BUILD BOX**[2.3.], **SONG BUILD BOX**[2.4.] or **CYCLER BOX**[**CYCLER**] puts an edit cursor into the clicked field (if editable).

b) Double left-click

Some functions can be launched by two succeeding short left-clicks over an object. for example: If you left-click over any source or destination entry in a **SELECTOR** , **BUILD**- or the **CYCLER BOX** you select the clicked item (it becomes reverse). Double-clicking over a source item *INSERTS* it into the destination window.

You can also select a source item in one of the above boxes by clicking once, enter a search string and launch the search with a double-click.

c) Single left-click while holding down the right mouse button

A few buttons provide more than one function.

If you left-click over the <*SYS/SPEED*> button the current **BEAT** is switched to **INTERNAL/SYSTEM SPEED**[3.5.5.]. Holding down the right button while doing this switches the **BEAT** to **INTERNAL SPEED** and takes over the current **SYSTEM SPEED** value.

Single-click over any of the **BEAT GRID POSITION**s fills/clears the position - doing the same while holding down the right button increases the **VELOCITY** of the current position (if filled).

d) Single left-click while holding down the <*ALTERNATE*> key on the ST keyboard

You can select single **SOUND CHANNEL**s by left-clicking over the respective display field in the **CHANNEL INFO**[3.2.]. Doing the same while holding down <ALTERNATE> selects/de-selects all sixteen **SOUND CHANNEL**s.

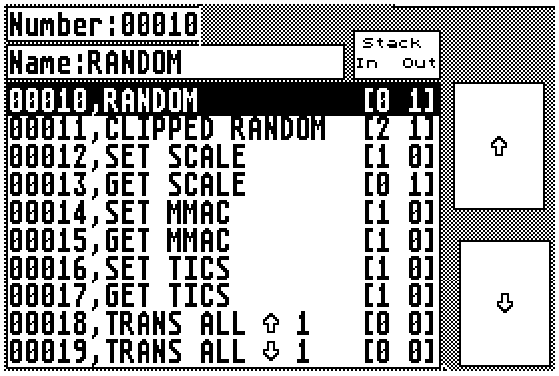
And holding down the <*ALTERNATE*> key on the ST keyboard while you left-click over a filled **BEAT GRID POSITION** brings up the **VELOCITY BOX** to change the velocity of the position.

e) Single left-click while holding down the <*CONTROL*> key on the ST keyboard

If you click over one of the display fields in the **CHANNEL INFO**[3.2.] while holding down the <*CONTROL*> key you can reverse the selection status of all sixteen S**OUND CHANNEL**s at once.

1.3. General Instructions for the SELECTOR-, BUILD- and CYCLER boxes

In this portion of the **CYCLER BOX** you see some multi-function buttons.



```
Number:00010
Name:RANDOM                  Stack
                             In  Out
00010,RANDOM                 [0 1]
00011,CLIPPED RANDOM         [2 1]
00012,SET SCALE              [1 0]
00013,GET SCALE              [0 1]
00014,SET MMAC               [1 0]
00015,GET MMAC               [1 0]
00016,SET TICS               [1 0]
00017,GET TICS               [1 0]
00018,TRANS ALL ⇧ 1          [0 0]
00019,TRANS ALL ⇩ 1          [0 0]
```

Number field:
Bring the cursor into this field by left-clicking on it.

Use the <*ESC*>, <*BACKSPACE*>, <*DELETE*> and <*CURSOR*> -keys to enter and edit text in this field. Numbers can be entered as decimals (=>53) or hexadecimals (=>$F4) using a ' $' - character header. Binary numbers (=>%01010) signed with a ' %' - character are also accepted. Launch a search for the entered number by double-clicking over this field.

Although you can enter numbers using decimal, hexadecimal and binary format the PATTERNER always displays numbers in decimal mode.

Name field:
Single-click over the name field to bring the cursor into edit position. Refer to the *Number s* description on how to enter and edit text. Double-click to launch a search for the entered string. You don' t have to enter complete names, the search routine looks for the nearest match. The search routine does distinguish between upper and lower case characters.

Display Window:
In the display window you can always see a portion of all available items. Any of these can be selected by left-clicking over it. The number and name of the selected item is shown in the number and name fields above. You can also browse through the data using the arrow up/down buttons which move up or down through the list by one entry on single-click or jump to the first/last entry on double-click. Using the arrow buttons does not change the current selection.

2. ABOUT THE DIFFERENT COMPONENTS

In this chapter we will try to explain you everything about the different components and how they work together.

PATTERNER lets you edit data on two different edit access levels.

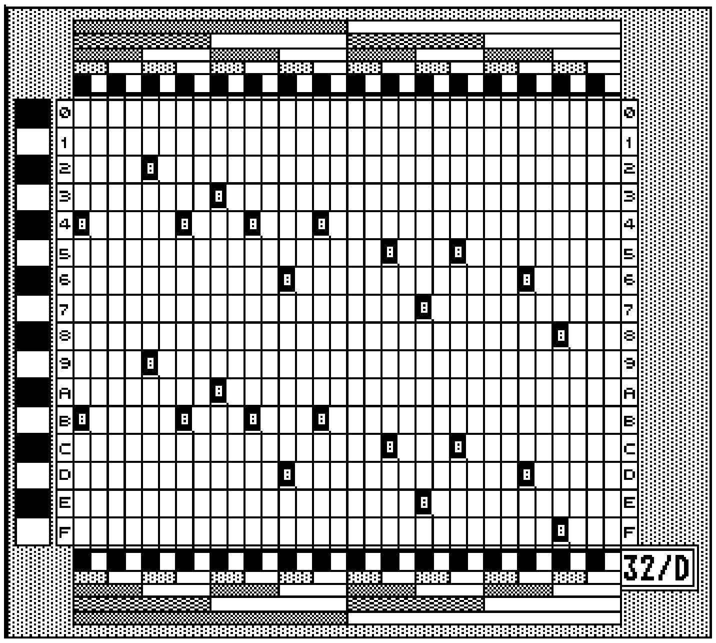
1) The **BANK** edit-level consists of the *SAVE/LOAD/EDIT* options in the *FILE* menu[4.2.] for each so called **BANK**. With these functions you can create new bank entries, change the names of existing entries, delete entries and load/save complete **BANKS**. However, in this level you are unable to change particular data WITHIN a **BANK** entry.

2) The **DATA** edit-level consists of the different **EDITOR boxes - PATTERN Editor Box**, **SONG Editor Box**, **SCALE Editor Box**, **MIDI Macro Compiler** and **BEAT GRID EDITOR**. Here you have access to the actual contents of particular **BANK** entries. But you can' t delete or rename bank entries or save/load **BANKS** in this level.

2.1. BEAT GRID EDITOR

The core of the system is the **BEAT GRID EDITOR**. This holds indiviual MIDI Note On/Off commands for each of the sixteen **SOUND CHANNEL**s (vertical, 0-F) through 32 **TIC**s (horizontal, 1-32). Note On messages are represented by filled grid positions while Note Offs are displayed as clear grid positions. Note On messages also hold velocity data which is displayed by numbers from 0 (minimum) to M (maximum) within each Note On for the monochrome version. In the color version three different colors (black, green and red) and five different fill heights are used to display velocities. The maximum velocity level is displayed with all three colors in one position. See also **VELOCITY BOX**[4.5.1.].

Click the left mouse button over an empty position to fill it, and to set its velocity to D**EFAULT VELOCITY**[4.5.1.]. Click the left mouse button over a filled position to clear it. You can change the **VELOCITY** of a filled position if you click the left mouse button over it while holding down the right mouse button. It is possible to hold both buttons down to achieve continous increase.



Another way to change the **VELOCITY** of one single filled **GRID POSITION** is to left-click the <*ALTERNATE*> key. This opens up the **VELOCITY BOX** where the current velocity of the position is shown. Exit this box by clicking over the button with the new desired velocity (if no change is wanted, exit with the selected button) [4.5.1.].

During playback of a **BEAT** the **TIC**s are scanned continously from left to right producing one MIDI Note On or Note Off message for each of the sixteen **SOUND CHANNEL**s contained in the **TIC**. All sixteen messages of a particular **TIC** are put out through the MIDI port. The playback routine looks up the note numbers in the **SCALE** assigned to the **BEAT**.

Additional data is stored in a **BEAT** structure and displayed within the **GRID EDITOR** window.

The top info textline above the **GRID** displays the following information:

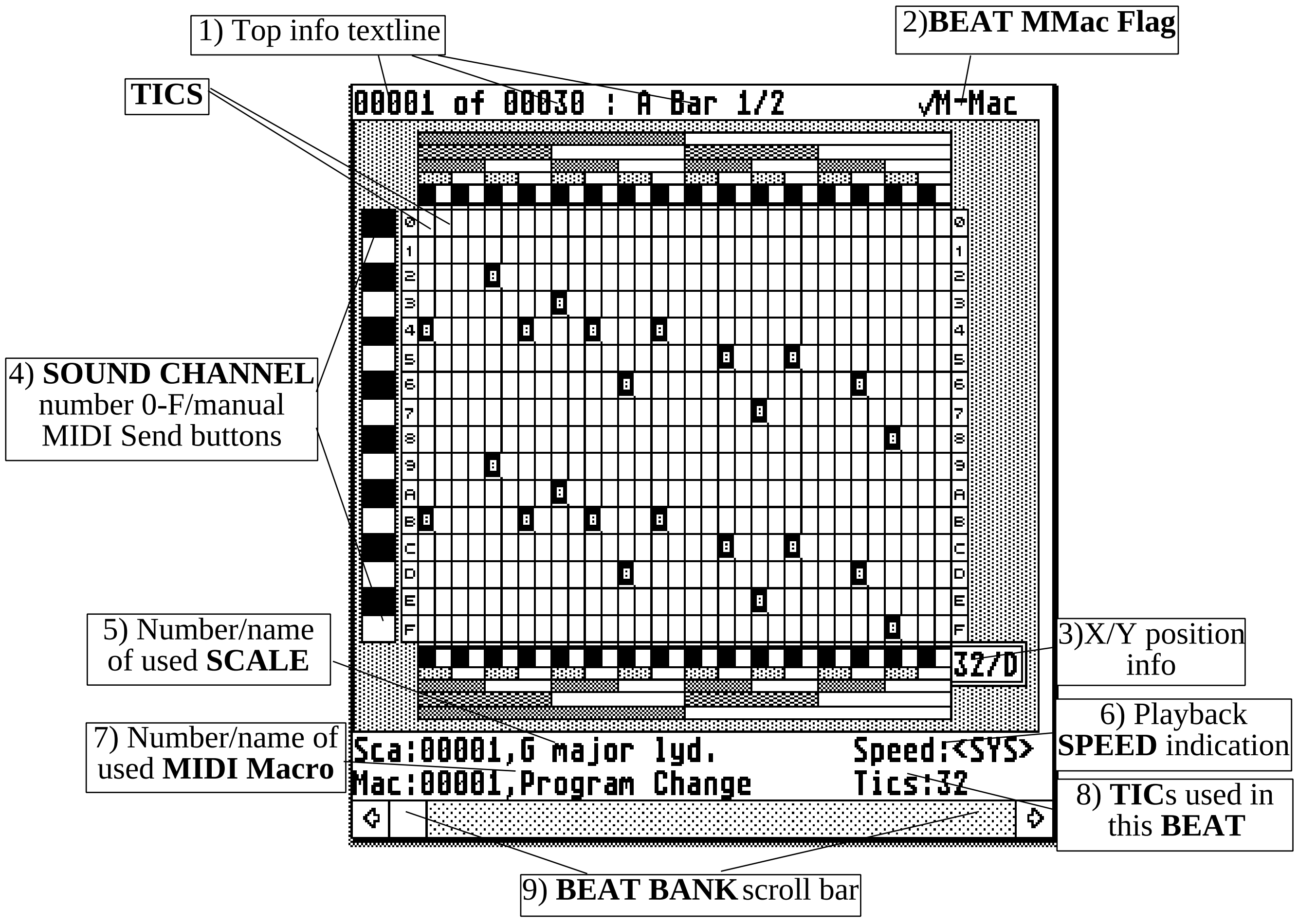
1) Number of the current **BEAT**. (Position within the **BEAT BANK**). Overall number of **BEATs** within the current **BEAT BANK**. Name of the current **BEAT**.

Single left-click over any of the above mentioned textinfos to open the **B**EAT BANK Editor[4.2.2.] or press <*CONTROL B*> on the keyboard.

2) **BEAT MMac flag**. This determines if the **BEAT** is enabled to carry out **MIDI Macros** (if ticked) or if the **BEAT**'s **MIDI Macro** is bypassed. To actually carry out **MIDI Macros** both, **BEAT MMac flag** and **global Enable MIDI Macro flag** [*FLAGS* menu 4.4.] must be set. Disable/enable the **BEAT MMac flag** by pushing the <*INSERT*> key on the keyboard or by left-clicking over this display.

3) X/Y position info. This little window becomes active whenever you move your mouse pointer over the **EDIT GRID**. Here are displayed the **TIC** number (1-32, left to right) and the **SOUND CHANNEL** number (0-F, top to bottom) of the actual pointer position.

4) **SOUND CHANNEL**s 0-F/manual MIDI Send Buttons. These are 16 checker-board buttons arranged vertically to the left of the **EDIT GRID**. Use these buttons to send single Note On messages to the receiving MIDI device. The MIDI keynumber (pitch) and MIDI channel are determined by the **SCALE** the current **BEAT** is assigned to. This function can be used to check MIDI keynumbers and MIDI channels of particular **SOUND CHANNEL**s or to simply play along with the running **BEAT**, **PATTERN** or **SONG**. The Note On message is repeated as long as you hold down the left mouse button over one of the Send buttons. The Note On uses the **DEFAULT VELOCITY** [4.5.1.]. With many sounds this causes stuck notes because no Note Offs are sent - press <M> on the keyboard or select *MIDI Shut Up* from the *FLAGS* menu [4.5.] to send a global All Notes off message.


1) Top info textline
2)BEAT MMac Flag
TICS
00001 of 00030 : A Bar 1/2
√M-Mac
4) SOUND CHANNEL number 0-F/manual MIDI Send buttons
5) Number/name of used SCALE
3)X/Y position info
32/D
6) Playback SPEED indication
7) Number/name of used MIDI Macro
8) TICs used in this BEAT
9) BEAT BANK scroll bar

Sca:00001,G major lyd. Speed:<SYS>
Mac:00001,Program Change Tics:32

The bottom info textlines below the **EDIT GRID** are displaying the following information:

5) Number and name of the **SCALE** this **BEAT** is assigned to.

6) **BEAT** playback **SPEED** indication. This is either <SYS> which means that the **BEAT** uses the **SYSTEM SPEED** shown in the **Playback Menu** or a value between 1 and 255 which is then used as playback **SPEED** by this **BEAT**[3.5.4./3.5.5.].

7) Number and name of the **MIDI Macro** this **BEAT** is assigned to. (This has no effect unless both the **Enable MIDI Macro flag** in *DEFAULTS* menu and the **BEAT's MMac flag** are checked - see top info textline).

8) Number of **TICs** which are actually played back in this **BEAT**. You can switch off a **BEAT** without deleting it from a **PATTERN** structure by setting this value to zero (0). Playback always scans the **TICs** from left to right. Click over this display or press <*ALTERNATE T*> to bring up the **NEW BEAT Box** where you can change the **TIC** value of the current **BEAT**[more on that box in 4.2.2.].

9) Use the scroll bar at the bottom of the **GRID EDIT WINDOW** to move through the **BEAT BANK** in both directions. Click either the *left* or *right arrow* button to move backward or forward one **BEAT**. Drag the scrollbox (un-filled section of the scroll bar) to move further distances through the bank. Like in other GEM programs the position of the scrollbox represents the relative position of the displayed data within overall data. In this case the position of the current **BEAT** within the complete **BEAT BANK**.

2.2. SCALES

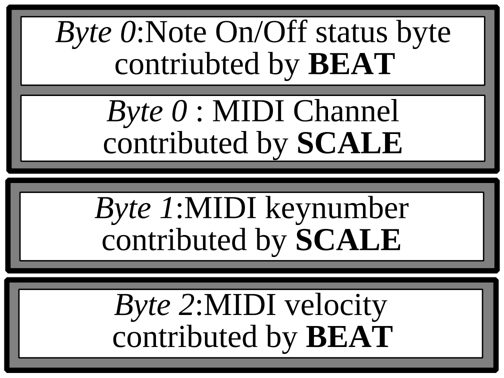
While **BEATs** deliver time and velocity data for the outgoing MIDI messages, **SCALEs** contribute pitches and MIDI channels. Every **BEAT** is assigned to one **SCALE** structure contained in the **SCALE BANK**. And thus, every **SOUND CHANNEL** of a **BEAT** will use the MIDI keynumber and the MIDI channel of the corresponding **CHANNEL** from the **SCALE** it is assigned to.

When the playback routine scans through the **BEAT's TICs** it assembles outgoing MIDI Note On/Off messages using the information contained in the **BEAT** structure and its assigned **SCALE**. Filled **EDIT GRID** position result in a Note on message which is composed of these three parts:

Byte 0 : Note On/Off status byte
Filled **GRID POSITIONs** yield to a Note On message while cleared **POSITIONS** create Note Offs. Note Off creation can be disabled or enabled using the NOTE OFF Flag in FLAGS menu [4.4.2.]. Another part of this byte carries the information about the message' s MIDI channel. At playback time this is looked up in the **SCALE**. MIDI Channels are set in the **SCALE Editor Box**.

Byte 1 : MIDI keynumber
This byte tells the receiving MIDI device which MIDI keynumber (pitch) is addressed. Use the **SCALE Editor Box** to define the MIDI keynumber for each of the **SCALE**'s sixteen **CHANNELs**.

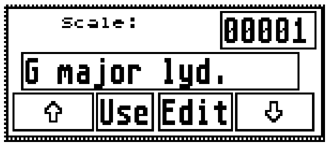
Byte 2 : MIDI velocity
This byte defines the velocity of the note. VELOCITY values are looked up in the **EDIT GRID**[2.1.] at playback time. Consult the previous chapter for details on **VELOCITY** editing.


Byte 0:Note On/Off status byte contriubted by **BEAT**
Byte 0 : MIDI Channel contributed by **SCALE**
Byte 1:MIDI keynumber contributed by **SCALE**
Byte 2:MIDI velocity contributed by **BEAT**

You find detailed information about MIDI messages and their formats in *APPENDIX D* or better, yet, in the manual(s) for the MIDI gear you use.

As mentioned earlier, global work within the **SCALE BANK** is done using the **SCALE BANK Editor** [4.2.5.]. To edit the contents of any **SCALE** you have to open the **SCALE Editor Box**. This can be done from the **MAINPAGE** by left-clicking the *EDIT* button in the **SCALE/MIDI Macro Switch Box**[3.4.] or by pressing <*CONTROL A*> on the ST keyboard.

This is how the **SCALE/MIDI Macro Switch box** initially looks like:



These are the things that you find in the **SCALE Editor Box**:

1) Channel Selectors

Select (activate) individual **SOUND CHANNEL**s by clicking upon their respective **CHANNEL** button. Selected **CHANNELs** are displayed in reverse mode. Only selected **CHANNELs** are transposed or able to receive MIDI In data from a connected keyboard. In addition many other functions use the **CHANNEL SELECTION PATTERN** to determine which channels to work on.

Besides individual selecting there are two group selection options:

A) Reverse the current **CHANNEL SELECTION PATTERN**
Click over any one of the **CHANNEL SELECTORs** while holding down the <*CONTROL*> key on the ST keyboard.

B) Select/un-select all **CHANNELs**
Click over any one of the **CHANNEL SELECTORs** and keep the <*ALTERNATE*> key on the keyboard pressed.

2) MIDI keynumbers

Next to each **CHANNEL SELECTOR** button you find a display of the MIDI keynumber, the note' s name and its octave. There are basically two ways to set the keynumbers:

A) Use the **SCALE EDIT** buttons (R*ESET*, *SCROLL UP/DOWN*, *TRANSPOSE UP/DOWN*) in addition to the **CHANNEL SELECTOR** buttons to adjust each **CHANNEL** to the desired keynumber. But that' s a lot more work than this second option:

B) Hook up the MIDI Out of your MIDI keyboard to the MIDI In of your ST. Now, press a key on the MIDI keyboard and all selected **SOUND CHANNELs** are set to the received MIDI keynumber. This option receives messages in OMNI mode (i.e. the MIDI Channel of the incoming message is ignored).

3) MIDI channels

Right to each of the sixteen keynumber displays you find a MIDI Channel info. MIDI channels can be set individually for each **SOUND CHANNEL** in the usual range of 1-16. To set MIDI Channels use the small arrow buttons above the MIDI Channel row. Only the MIDI channels of selected **SOUND CHANNELs** are affected. This is one of the exceptions where PATTERNER uses values starting at number one (1) instead of zero (0). The main reason for this exception is the fact that all MIDI gear I encountered treated MIDI Channels from 1-16 instead of 0-15. However, keep in mind that internally PATTERNER still looks at MIDI Channels 1-16 as 0-15 (important in **MIDI Macros** and ***CYCLER***).

4) User text fields

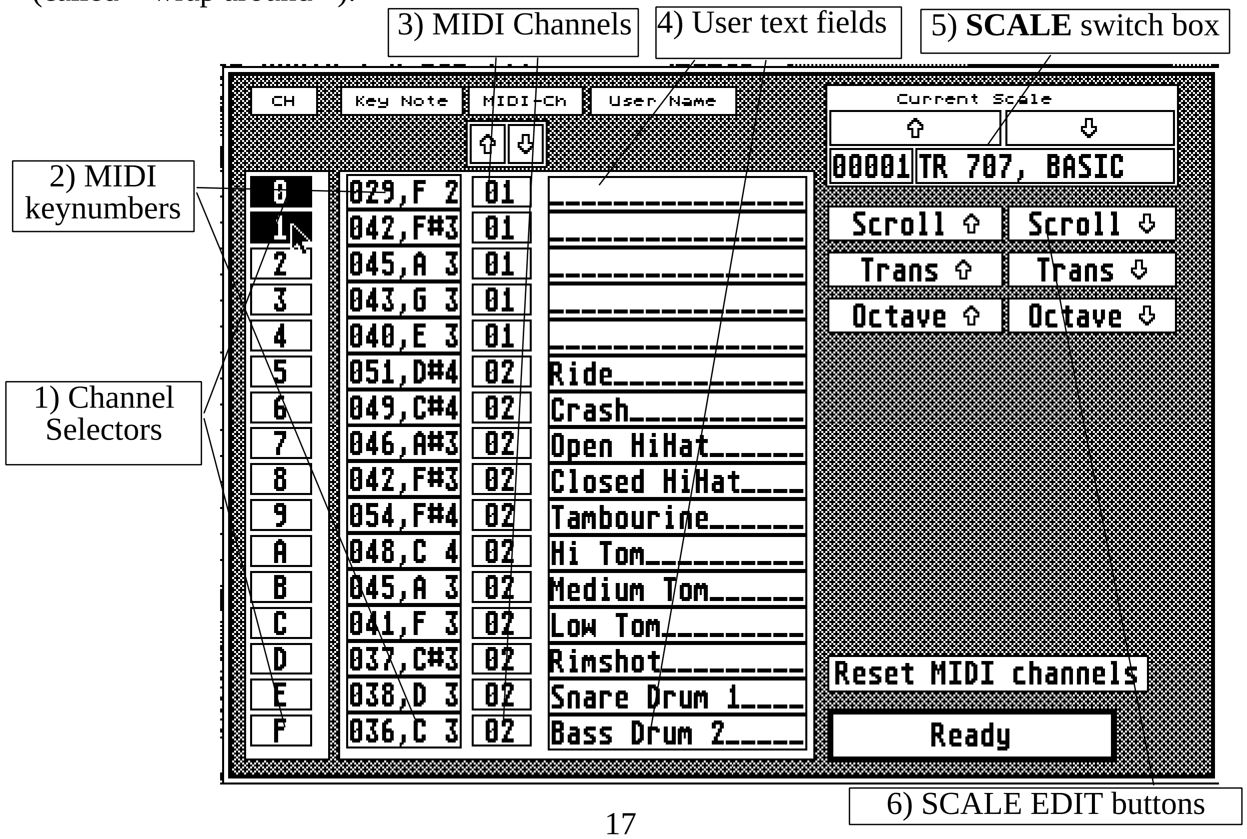
Right to each MIDI Channel display you see a sixteen character text field. Whatever information you put in there is ignored on playback. But it makes sense to enter comments about the **SCALE**, or the function of a single **CHANNEL** (i.e. If you set up a **SCALE** to use with a drum sound patch enter the instruments' names and other information). A sized down copy of the User text fields can be displayed on the **MAIN PAGE**. See *Extended CHANNEL INFO* [3.3.2.] on how to do that.

If you want to enter text into any of the User text fields you must switch over to Text Mode. To do this click over the *USER NAME* button. Now you can access the sixteen text fields by using the <*CRSR*> keys on the ST keyboard or by mouse clicking. You can edit the text using all your regular edit keys[1.3.]. Press <*RETURN*> on the ST keyboard or click the *READY* button to exit Text Mode. While in Text Mode you have no access to any other objects than the Text fields and the *READY* button.

5) **SCALE** switch box

You can select any other **SCALE** from the current **SCALE BANK** within the **SCALE Editor Box**. Use the **SCALE** switch box in the upper/right corner of the **SCALE Editor Box**, to move up/down through the **BANK**.

If the **SCALE** switch box displays the last entry of the **SCALE BANK** the up arrow will get you back to the first entry. Moving downwards from the first entry gets you to the last entry (called 'wrap around').



00001 TR 707, BASIC

CH	Key Note	MIDI-Ch	User Name
0	029,F 2	01	________________
1	042,F#3	01	________________
2	045,A 3	01	________________
3	043,G 3	01	________________
4	040,E 3	01	________________
5	051,D#4	02	Ride____________
6	049,C#4	02	Crash___________
7	046,A#3	02	Open HiHat______
8	042,F#3	02	Closed HiHat____
9	054,F#4	02	Tambourine______
A	048,C 4	02	Hi Tom__________
B	045,A 3	02	Medium Tom______
C	041,F 3	02	Low Tom_________
D	037,C#3	02	Rimshot_________
E	038,D 3	02	Snare Drum 1____
F	036,C 3	02	Bass Drum 2_____

6) Just below the **SCALE** switch box you find a set of *SCALE EDIT* buttons.

A) The *SCROLL UP* button rotates all **CHANNEL** contents (MIDI keynumber, MIDI Channel and Text Fields) up one notch. The one entry which is moved out at the top is inserted as the new entry at the bottom. The contents of CHANNEL 1will move to CHANNEL 0, CHANNEL 2 to CHANNEL 1, etc...CHANNEL 0 will move to CHANNEL F.

B) *SCROLL DOWN* has the same effect only in downward direction. CHANNEL F will move to CHANNEL 0.

Both *SCROLL*ing functions are not affected by the **CHANNEL SELECTION** buttons.

C) *TRANSPOSE UP* performs a halfstep up transposition on every selected **CHANNEL**. **CHANNELs** containing a keynumber of 127 are reset to 0.

B) *TRANSPOSE DOWN* transposes all selected **CHANNELS** down one halfstep. **CHANNELs** containg a keynumber of 0 are set to 127.

C) *OCTAVE UP*performs an octave up transposition on every selected **CHANNEL**.

D) *OCTAVE DOWN* transposes all selected **CHANNELs** down one octave.

When you use *OCTAVE TRANSPOSITION* be aware that transposing up from the highest octave (shown as ' :') you will come back into the lowest octave (' 0'). If you transpose down an octave from the lowest octave range (' 0') pitches are taken into the highest octave (' :'). Also in both directions the basic pitches will change (a ' C' will not stay a ' C') in this case.

- Use the *RESET MIDI CHANNELS* button to reset selected MIDI Channels to channel 1

- Click the *READY* button to exit the **SCALE Editor Box**.

For further options in **SCALE EDIT**ing see **SCALE MICRO COMMANDS** [4.5.4.].

2.3. PATTERNS

So far we have covered the very basics of the PATTERNER: The system which creates MIDI Note On/Off messages by combining data from **BEATs** and **SCALEs**. As you know PATTERNER lets you edit in real-time so you can switch **BEATs** on the **MAINPAGE** while playback is running. This is nice to experiment with certain things. However, the next step is to arrange your BEATs in a playback list called **PATTERN**. A **PATTERN** can best be compared to a measure in regular musical terms, although its structure can be much more complex.

A **PATTERN** is composed of a sequence of **BEATs** arranged in a certain order. Playing back a **PATTERN** means to play back the **BEATs** contained in the **PATTERN** in the sequence they were programmed into the **PATTERN** structure. After a **PATTERN** has been played back through all of its contents it either starts from its beginning or gives control to the calling structure: A **SONG** [2.4.].

PATTERNs are assembled and editied in the **PATTERN Editor Box** [*EDIT PATTERN* on **MAINPAGE**, 3.1.3.] while global work in the **PATTERN BANK**, such as creating and deleting entries is done using the *PATTERN EDIT* option from *FILE MENU* [4.2.3.].

SONGS

Although **PATTERNs** can be quite big (containing up to ~65500 **BEATs** if your RAM holds that much) you might still want to use another level of organization. This level consists of one or more **SONGs**, all contained in the **SONG BANK**. A **SONG** is created and edited exactly the same way like a **PATTERN** only you do it in the **SONG Editor Box** and use **PATTERNs** instead of **BEATs**.

Musically a **SONG** contains all the sections and parts of a performance which are represented in the individual **PATTERNs**. Look at the common musical form AABA, where you usually have two similiar eight measure A-sections, one eight measure B-section (or bridge) and another A-section. In this case I would put all **BEATs** of the A-section into a **PATTERN** called "A-section" and all B-section **BEATs** into a **PATTERN** "B-section". Now put together a **SONG** containing 2x"A-section", 1x"B-section" and 1x"A-section". This is basically how the **BEAT/PATTERN/SONG** structures are used.

SONGs are assembled and edited in the **SONG Editor Box** [*EDIT SONG* on the **MAINPAGE**, 3.1.4.].

2.4. MIDI Macros

Unlike **SCALEs** and **BEATs, MIDI Macros** are not necessarily needed to create a performance. However, the more options your MIDI gear has the more you'll like them. Since a **MIDI Macro** can contain totally user definable data you can use it for just about everything. A few examples:

Switching sound patches/programs
Send additional notes or chords
Send controller messages (pitch bend, modulation stuff...)

2.4.1. **MIDI Macros** launched by a **BEAT**

Every **BEAT** is assigned to one **SCALE** and to one **MIDI Macro**. While the contents of the **SCALE** are used all the time to look up MIDI keynumbers and MIDI Channels, the bytes from the **MIDI Macro** are sent out to the ST's MIDI out port before the first **TIC** of the **BEAT** is scanned. This means that a **MIDI Macro** which switches MIDI programs/patches does this right before the **BEAT** is played. In this mode **MIDI Macros** are connected to **BEATs** only and have nothing to do with **PATTERNs** or **SONGs** (i.e. it doesn't matter if the **BEAT** is played by itself or called from a **PATTERN** or **SONG**).

In order to launch a **MIDI Macro** from a **BEAT** you must enable two flags:

1) Global **MIDI Macro Enable flag** [*FLAGS* menu 4.4.1.]
This flag enables or disables the global launch of **MIDI Macros** from any **BEAT**. Toggle this flag by clicking *Enable MIDI Macros* in the GEM *FLAGS* menu.

2) BEAT **MMac flag** [**BEAT GRID EDITOR** 2.1.]
While the previous flag affects every **BEAT** the **MMac flag** enables/disables individual **BEATs** to launch their **MIDI Macros**. However, keep in mind that both flags need to be enabled to use **MIDI Macros** from a **BEAT**. To toggle the **BEAT MMac flag** use the <*INSERT*> key on the ST keyboard or left-click over the flag's display in the **GRID EDITOR**.

Now, that was the first way to launch **MIDI Macros**.

2.4.2. **MIDI Macros** launched from the ST's number pad

The second way to launch a **MIDI Macro** is by manual control. This works in real-time and can be used to execute **MIDI Macros** independent from the running performance. I like to use this to remote control my MIDI gear.

To use this feature you have to assign the needed **MIDI Macros** from your **MIDI Macro BANK** to the keys of the ST's numberpad. This is done in the **MIDI Macro Key Box** [4.5.2.].

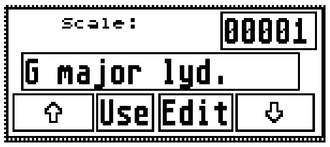
After assigning one or more **MIDI Macros** to the numberpad keys 0-9 the respective **MIDI Macros** can be called up anytime from the **MAINPAGE** by pressing the respective key on the ST keyboard or by using the numberpad buttons on the **MAINPAGE** [3].

There is of course a third possibility to launch **MIDI Macros**: from within **CYCLEs**. But we won't go into that here - look it up in the **CYCLER** part of this book.

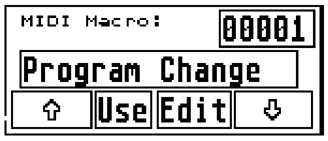
2.4.3. **MIDI Macro Compiler Box**

And this is where you create and edit **MIDI Macros**: the **MIDI Macro Compiler Box**. Open up this dialog box fom the **MAINPAGE's SCALE/MIDI Macro Switch Box**. If this **Switch Box** looks like the one here carry out step 1:

This is the **MAINPAGE SCALE/MIDI Macro Switch Box** in **SCALE** mode. All buttons clicked in this mode affect the **SCALE BANK**/current **SCALE**. If this box is already in **MIDI Macro** mode skip step 1.



Step 1: Single-left click over 'Scale' in this box or press <*ALTERNATE A*> on the ST keyboard to switch into **MIDI Macro** mode. After this the **Switch Box** looks like that:



You are now in **MIDI Macro** mode and all button clicks affect the **MIDI Macro BANK**/current **MIDI Macro**.

Step 2: Left-click over the *EDIT* button or press <*ALTERNATE E*> on the ST keyboard.

By now you know that you have to use the **MIDI Macro BANK Editor**[4.2.6.] to create new **BANK** entries, rename or delete them. However, the contents of individual **MIDI Macros** are edited in the GEM box explained in the following.

To make data entry as flexible as possible you enter all data by typing instead of clicking buttons. As soon as the **MIDI Macro Compiler** pops up the data of the current **MIDI Macro** are displayed in the editor. If there is no data within the **Macro** the editor stays empty (naturally).

Like in the **SCALE Editor Box** use the up/down arrow buttons in the '*Current MIDI Macro*' box to scroll up/down through the **MIDI Macro Bank**.

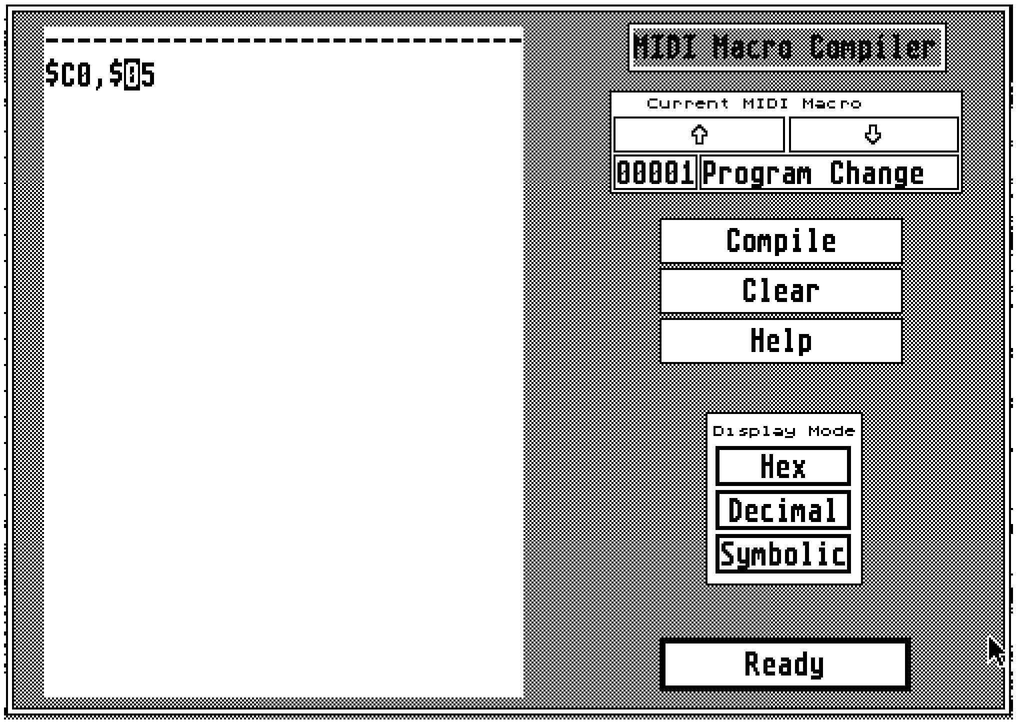
Only characters that can be processed by the compiler are accepted as input:$% ,;_0123456789ABCDEFabcdef. All others are ignored.

The **Compiler** recognizes these three number formats:

1) Hexadecimals (using a '$' -header as in '*$f4*')

2) Decimals (no header needed as in '*244*')

3) Binaries (with a '*%*' -header like in '*%11110101*')



```
-------------------------------
$C0,$05
```

COMPILE

The compiler creates a **MIDI Macro** out of all ASCII data in the edit window. There are no error messages such as out of range errors or the like. All inputs are compiled into something. When you use wrong number formats (for example %109=> there IS no ' 9' in binary) the compiler WILL create data anyway. Numbers such as $f34 or 457 are truncated into values below 255/$ff (in this case $f34 becomes $34/52 and 457 becomes $c9/20; this indicates clearly that only bits 0-7 of a value are relevant). When the compiler has finished its work all MIDI commands (those with a set bit 7, all numbers above $7f/127) start in a new row to denote them as MIDI status bytes followed in the same line by their data bytes (if any).

The current editor screen is also compiled if you:

1) Scroll up/down through the **MIDI Macro BANK**

2) Exit the **MIDI Macro Compiler** by clicking the *READY* button.

Pressing <*ESC*> on the ST keyboard has the same effect like left-clicking over *COMPILE*. This operation is not undo-able!

CLEAR

This clears the edit screen. However, until the *COMPILE* button is used after clearing the screen, the current **MIDI Macro** is not touched and may be recalled by pressing the *UNDO* key on the ST keyboard.

Pressing <*SHIFT HOME*> on the ST keyboard has the same effect.

HELP

Here you can call up a help screen which displays all available keyboard commands for the editor. This display can only be left by pressing any button on the keyboard (NOT clicking with the mouse!). This is not undo-able!

You can also access this function by pressing the <*HELP*> key on the ST keyboard.

HEX

This displays all data in the edit window in hexadecimal mode (' $' character in front of all numbers).

You can also switch to hexadecimal output by pressing the <*F2*> key on the ST keyboard.

DECIMAL

Displays all data in the edit window in decimal mode (no headers).

The <*F3*> key on the ST keyboard has the same function.

SYMBOLIC (not 1.3)

READY

Exit the **MIDI Macro Compiler**. Before the box is closed the current edit screen contents are compiled into the current **MIDI Macro**. This is not undo-able!

Here a list of all available keyboard commands for the **MIDI Macro Compiler**:

<*CRSR RIGHT*>

Move edit cursor one column to the right. If the cursor is in the rightmost column it jumps to the first column in the next row.

<*CRSR LEFT*>

Move the edit cursor one column to the left. The cursor moves to the rightmost column one row above if already in first column.

<*CRSR UP*>

Moves cursor one row up. If in first row, no further moves upward.

<*CRSR DOWN*>

Moves cursor down one row. If already in last row, no further downward moves.

<*BACKSPACE*>

Moves the cursor, the character under the cursor and with it all characters right to it one column to the left. Affects the current row only.

<*RETURN*>

This moves the cursor into the next row first column.

<*ENTER*>

This inserts an empty row at the current cursor position. The contents of the very last row are scrolled out of the editor, so be carefull!

<HOME>

The cursor moves to its home position, first row, first column.

<ESC>

Starts the **COMPILER**. This is not UNDO-able!

<SHIFT HOME>

This moves the cursor into the left upper edit window corner and clears the window from all text.

<UNDO>

Most of the editor commands can be UNDOne, as long as the screen has not been **COMPILED** (unless otherwise noted).

<DELETE>

This deletes the character under the cursor and moves all characters to its right one column to the left.

<INSERT>

Inserts a blank (SPACE) at the current cursor position, without moving the cursor.

<F1>

Deletes the current row and moves all rows below one row up.

<F2>

Data is displayed in hexadecimal.

<F3>

Data is displayed in decimal.

<HELP>

Brings up the help page, containing all the available keyboard commands. You can only exit this help screen by pushing any key on the ST keyboard. Before the help screen shows up all data in the editor is compiled. This is not undo-able!

To utilize the **MIDI Macro** features to their maximum you should learn as much about MIDI in general as you learn about the MIDI implementation of the MIDI gear you use. For further detail on general MIDI messages refer to *Appendix D*.

2.6. SYSTEMS

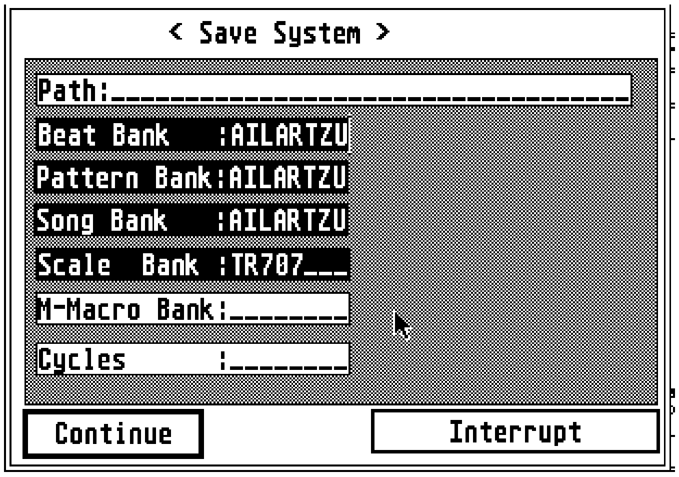
Until now we have been talking about the different components which contribute data to a PATTERNER performance. We covered **BEATs, SCALEs, PATTERNs, SONGs** and **MIDI Macros**. All these structures are organized in their respective **BANK** where they can be loaded from disk, saved or deleted (and created at that). You can save/load every **BANK** individually which makes a lot of sense if you are working in one single **BANK** while all others stay un-touched. However, the more experienced you become the more complex your performances will get.

Whenever you have worked on two or more **BANKs** which belong together (for example:a certain **BEAT BANK** and some **SCALEs** from the **SCALE BANK**) you should save them as a **SYSTEM**. Save and load **SYSTEMs** using the respective *FILE* menu entries [4.2.1.]. To save a **SYSTEM** you have to enter some stuff into the <*SAVE SYSTEM*> box:

1) The names of the different **BANKs** as you want them to be saved. This is done in the **BANK** fields. Enter names up to eight characters long. File extensions are handled by the save routine.

2) Only **BANKs** whose **BANK** field is selected will be saved. In the picture below all but the **MIDI Macro BANK** are saved.

3) Exit this box with *CONTINUE* if all adjustments are made. Or *INTERRUPT*.


< Save System >

Path:_________________________________

Beat Bank   :AILARTZU

Pattern Bank:AILARTZU

Song Bank   :AILARTZU

Scale Bank :TR707___

M-Macro Bank:________

Cycles   :________

In version 1.3 the Path and the CYCLE fields have no purpose.

A **SYSTEM** saves these files:

1) All **BANKs** which have been selected in the **SYSTEM** box.

2) A **SYSTEM** file with the names of the saved **BANKs**. This file also contains the status of all system flags (upper half of *FLAGS* menu[4.4.]) and the **SYSTEM SPEED**.

When you load a **SYSTEM** from disk the **SYSTEM box** pops up showing you which **BANKs** have been saved. The **BANK** field of every **BANK** contained in the **SYSTEM** will be selected and filled by a name. You don' t have to load all selected **BANKs** since you can un-select the ones you don' t need. All files of a **SYSTEM** have to be in the same folder or on the same path.

2.7. CYCLES

You know by now that PATTERNER gives you real-time control over just about anything. You can change all parameters of a **BEAT, SCALE** or whatever while playback is running. But listen to this:

Imagine the playback routine as the section of the PATTERNER which looks up stuff in the **BEATs, SCALEs, MIDI Macros** and other **BANKs** and throws it together to send it to the ST' s MIDI out. At four distinctive spots in this routine you can insert your own programs.

1) After every finished **TIC** of a **BEAT**

2) After every finished **BEAT**

3) After every finished **PATTERN**

4) And after every finished **SONG**

Naturally, the last two options can only be used if you playback a **PATTERN** or a **SONG**.

At these time spots you can change almost any parameter of a performance or within the **BANKs** which contribute data to its production. The funny thing is that not you do it but it is doing this automatically. Plus you can read parameters from the PATTERNER, process them in some way and write them back. Or use one parameter to change a couple of others. The ' mutating music system' .

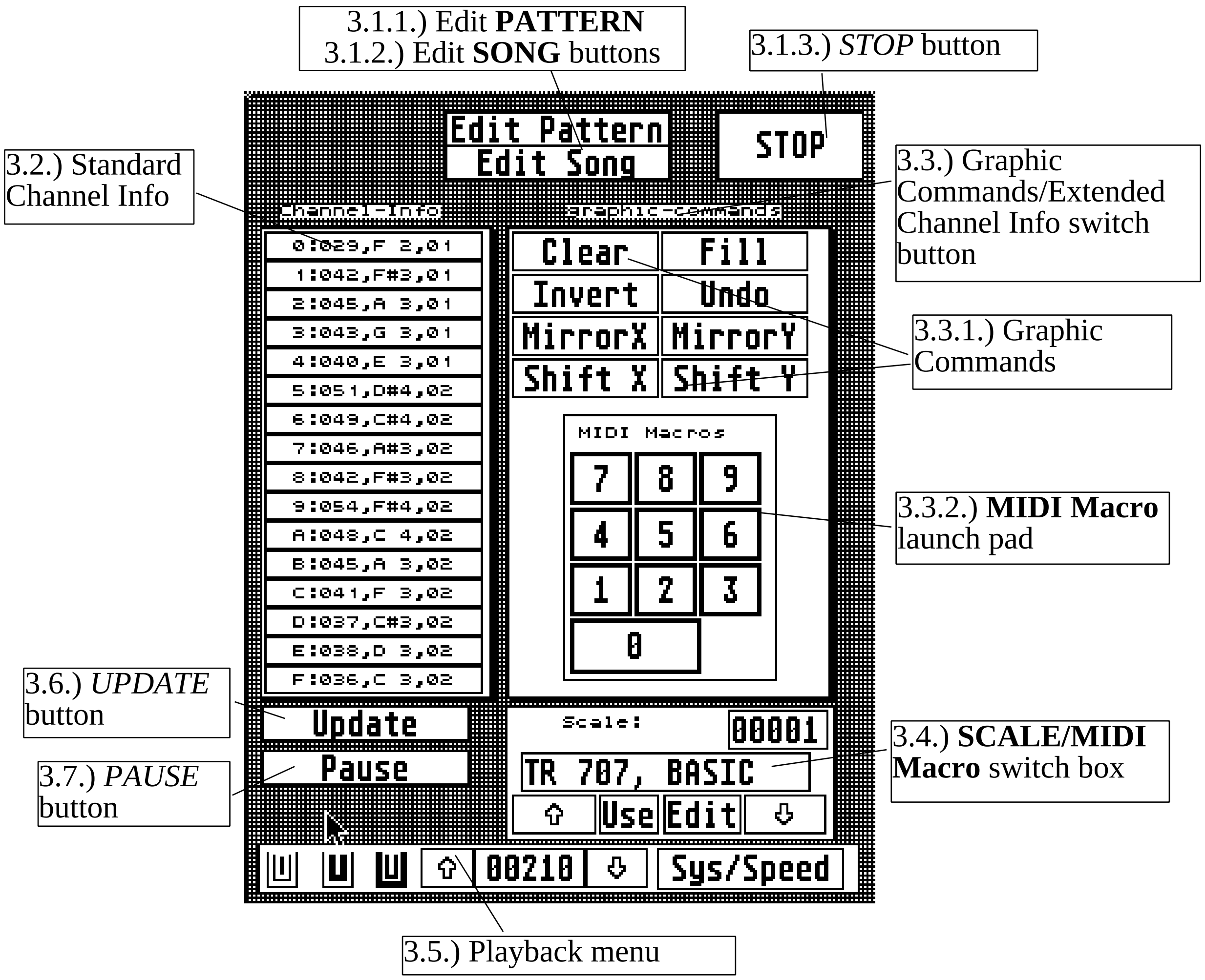
Those four ' user-definable' parts of the playback routine are called **CYCLES**, because they are run through in ' cyclic' movements (and because the name sounds interesting).

CYCLEs also give you access to an internal MIDI IN buffer. This buffer records ALL (or filtered) MIDI data coming in from the connected keyboard. Currently there is one routine in the ***CYCLER*** library which lets you read that buffer. As a relatively new addition to the program its potential can' t be overlooked yet. Other routines dealing with this buffer will follow in future updates.

Since **CYCLEs** are such a powerful and yet unexplored area they got their own portion of this book called **CYCLES** (what did you think?).

3. MAIN PAGE

First thing you' ll see after starting PATTERNER is the **MAINPAGE**. This section contains the most regularly used buttons and displays.



Since the **MAINPAGE** is the part of PATTERNER you' ll be using most often it provides the widest range of tools.

3.1.1. *EDIT PATTERN* button

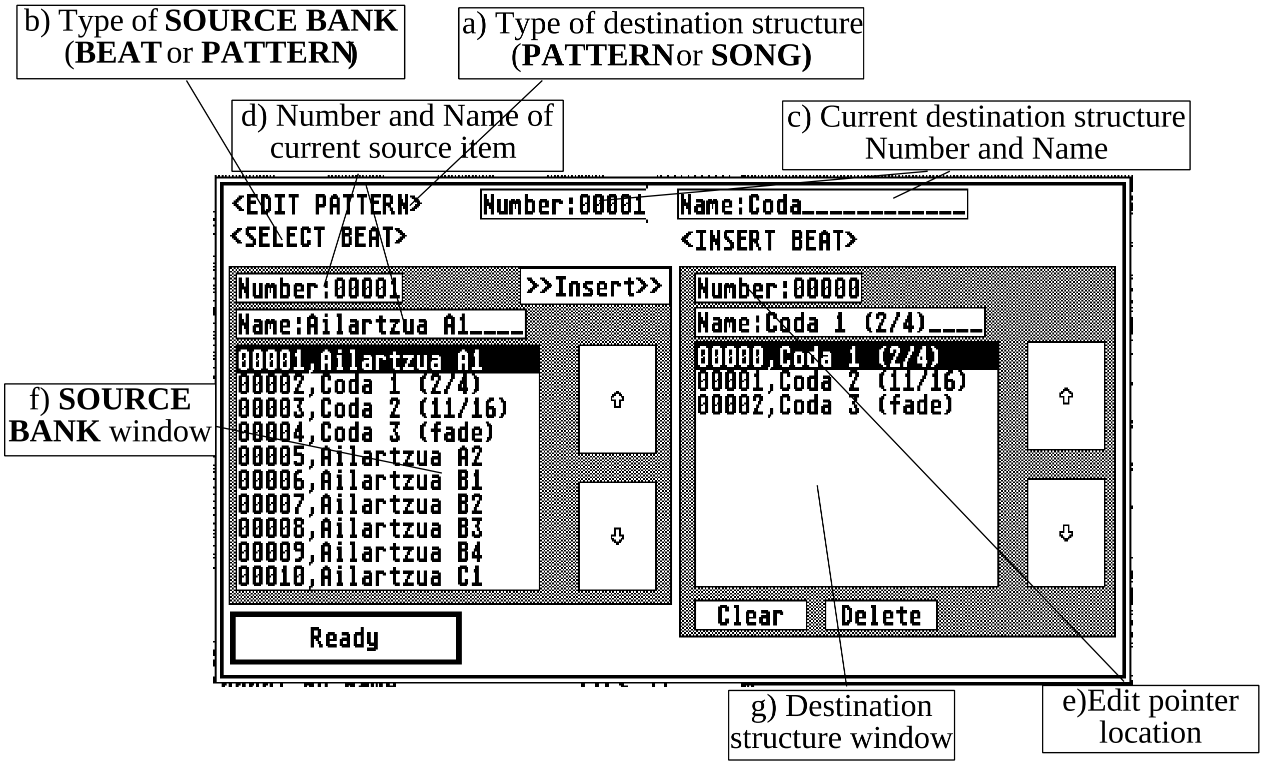
3.1.2. *EDIT SONG* button

Both of these options are working the same way. The only difference are **SOURCE BANK** and **Destination Structure**.

To build a **PATTERN** you use a destination **PATTERN** (contained in the **PATTERN BANK**) and the **BEAT BANK** as **SOURCE BANK**.

To build a **SONG** you use a destination **SONG** (from the **SONG BANK**) and the **PATTERN BANK** as **SOURCE BANK**.

This picture shows the **PATTERN EDITOR Box**


b) Type of SOURCE BANK (BEAT or PATTERN)
a) Type of destination structure (PATTERN or SONG)
d) Number and Name of current source item
c) Current destination structure Number and Name
f) SOURCE BANK window
g) Destination structure window
e)Edit pointer location
<EDIT PATTERN>
Number:00001
Name:Coda
<SELECT BEAT>
<INSERT BEAT>
Number:00001
>>Insert>>
Name:Ailartzua A1
Number:00000
Name:Coda 1 (2/4)
Clear
Delete
Ready

```
00001,Ailartzua A1
00002,Coda 1 (2/4)
00003,Coda 2 (11/16)
00004,Coda 3 (fade)
00005,Ailartzua A2
00006,Ailartzua B1
00007,Ailartzua B2
00008,Ailartzua B3
00009,Ailartzua B4
00010,Ailartzua C1
```

```
00000,Coda 1 (2/4)
00001,Coda 2 (11/16)
00002,Coda 3 (fade)
```

Here a description of the various buttons and displays you see in this box:

a) Type of destination structure

Since you use the same editor box to edit **PATTERNs** and **SONGs** you need this string to tell you what you are assembling.

b) Type of **SOURCE BANK**

If you are working on a **PATTERN** your building material will be **BEATs** contained in the **BEAT BANK**. While working on a **SONG** you draw from the **PATTERN BANK**

c) Current destination structure Number and Name

These are two text fields containing number and name of the current destination **PATTERN** or **SONG**. To switch to another **PATTERN/SONG** enter the number of the desired **BANK** entry into the number field and double-click over it. You can also enter the name of the requested entry into the name field and double-click over that. You don' t have to enter the complete name, the search routine looks for the nearest match. The found item becomes the new current selection.

d) Number and Name of current source item

These two textfields display the name/number of the currently selected source **BEAT/PATTERN** Both textfields can be used to search a particular entry by number or name. Use the same procedure of filling in number or name and double-clicking like mentioned in c).

e) Edit pointer location

These number and name fields display the position of the edit pointer within the destination structure. Clicking the *DELETE* button deletes this entry. INSERTING the currently selected **SOURCE** item will insert just in front of this position.

To set the edit pointer in the destination structure click upon the desired entry within the right selector window. Its number and name will be shown in the current edit pointer text fields and the selection in the window will be drawn reverse.

The number of the edit position is the position number within the structure whereas its name is taken from the **SOURCE BANK**. This means that if you load a different **SOURCE BANK** the name will change most likely.

If the structure is empty or the edit pointer is beyond the last entry the number field displays the next number after the last entry and the text field shows the string:E*ND OF STRUCTURE*.

The current edit position is displayed in reverse mode.

f) **SOURCE BANK** window

The source bank buttons and displays are used like those you find in the **BANK SELECTOR** windows [*BEAT BANK EDIT* 4.2.2.]:

1) The **SOURCE BANK** window displays a part of the **BANK** contents. Click over the desired entry to make it the current selection. The current selection appears in reverse video within the window and in the number/name field just above the window.

2) Double-clicking over a **SOURCE BANK** entry makes this selection the current selection and *INSERTs* it into the destination structure [see also *INSERT]*.

4) Use the scroll buttons to scroll up or down through the **SOURCE BANK**. This does NOT affect the current selection.

5) Double-click over the arrow buttons to scroll to first/last source list entry immediately. This does NOT affect the current selection.

g) Destination structure window

In this window you see a part of your already assembled **PATTERN** or **SONG** (or all of it if it isn't too large). Click over any visible entry to make it the current selection which is always displayed by number and name in the textfields just above the window. The current selection in this window represents the position of the edit pointer.

Use the arrow buttons in the same manner like described under f).

CLEAR

Clears the **SONG/PATTERN** structure completely. This is not reversible. After *CLEAR*ing the edit pointer number will be '1' and the name field will read '*END OF STRUCTURE*'.

DELETE

The entry at the edit pointer position (example: 00004,NAME) is deleted. All positions higher are moved one position. The pointer stays at the same position (00004).

An '*END OF STRUCTURE*' is not delete-able.

Use the *READY* button or press <*RETURN*> to exit the **PATTERN/SONG EDITOR Box** (in whatever mess - all changes are final!).

Also use <*ALTERNATE P*> to call up the **PATTERN Editor Box** and <*ALTERNATE S*> to open the **SONG Editor Box** from the **MAINPAGE**.

3.1.3. STOP button

The *STOP* button stops every playback sequence. No data is changed.

Use the <*ESC*> key to access the **STOP** function from the keyboard.

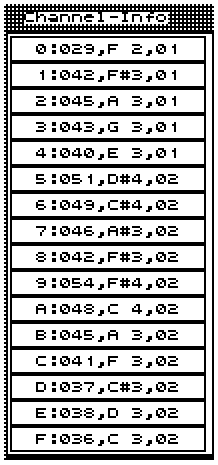
Conceptually the *STOP* button belongs to the commands in the playback menu but has been moved to another screen area to avoid accidentally touching it while fiddling with the *UPDATE* or *SPEED* buttons.

Like on most advanced cassette players you don't have to press *STOP* before starting another function. If you launch a **SONG** playback in the middle of a **BEAT** playback the **BEAT** just stops and the **SONG** starts.

In some cases when you work with the features of the **CYCLER** section the regular *STOP* function doesn't work. PATTERNER has a built in **EMERGENCY BRAKE** which is activated by pressing <*SHIFT LEFT/ALTERNATE*> on the keyboard. I won't go into details here because in basic operation mode you don't need it [more on this in *CYCLER LIBRARY* Reference, *CYCLER* 4.2. under *R_UPDATE*].

3.2. Standard Channel Info

These are sixteen function/display buttons corresponding with the sixteen **SOUND CHANNELs** of a **BEAT**. Displayed are **SOUND CHANNEL** number, the assigned MIDI keycode as the note' s name and octave in decimal numbers and the MIDI Channel.



```
Channel-Info
0:029,F 2,01
1:042,F#3,01
2:045,A 3,01
3:043,G 3,01
4:040,E 3,01
5:051,D#4,02
6:049,C#4,02
7:046,A#3,02
8:042,F#3,02
9:054,F#4,02
A:048,C 4,02
B:045,A 3,02
C:041,F 3,02
D:037,C#3,02
E:038,D 3,02
F:036,C 3,02
```

Any of these buttons can have two switch states:

1) The button drawn in normal video (black letters on white ground), when the corresponding **SOUND CHANNEL** is not selected (and not accessible for a lot of edit functions).

2) A button drawn reverse (white letters on black ground), displays that the corresponding **SOUND CHANNEL** is selected and now available for a lot of functions.

Left-clicking upon any of these buttons reverses its status.

Left-clicking over any button while holding down the <*CONTROL*> key inverts the current **CHANNEL SELECTION PATTERN**

Left-clicking over any button while holding down the <*ALTERNATE*> key selects/unselects all **CHANNELs**. (also see *SCALEs* [2.2.] for more information on C**HANNEL SELECTION**).

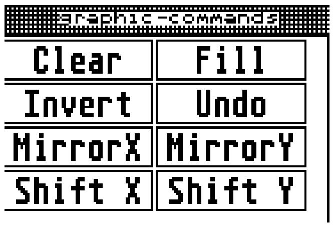
In general the **CHANNEL SELECTION** feature is used to clip editing such as **MICRO COMMANDS** or **GRAPHIC EDIT** commands to the selected **SOUND CHANNELs**. Many functions in the **SCALE EDITOR Box** [2.2.] are dependent on the selection pattern. Also many ***CYCLER*** programs are using the **CHANNEL SELECTORS** to interact with PATTERNER [*CYCLER*].

3.3. Graphic Commands/Extended Channel Info switch button

This text string serves as a display/switch button. Click on it to switch back and forth between '*Graphic Commands*' and an '*extended Channel Info*'. You can also switch back and forth by pressing *<ALTERNATE D>* on the computer keyboard.

3.3.1. Graphic Commands

This group of GEM buttons affects only the contents of the **EDIT GRID** (i.e.current **BEAT**). Some of them work on all **SOUND CHANNELs** of the current **BEAT** while others change only the selected **CHANNELs**.



CLEAR
This clears all selected **CHANNELs**.

FILL
This fills all selected **CHANNELs**. All filled positions are set to **DEFAULT VELOCITY** [4.5.1.].

INVERT
The fill pattern of all selected **SOUND CHANNELs** is reversed. Clear positions are filled and filled positions are cleared. All positions which are filled receive the **DEFAULT VELOCITY** [4.5.1.]

UNDO
This restores the **EDIT GRID** like it was just before the execution of the last **GRAPHIC COMMAND**. Only the last command is *UNDO*-able.

MIRRORX
This mirrors all **SOUND CHANNELS** in the **EDIT GRID** at the middle x-axis. **SOUND CHANNEL** 0 gets the contents of **CHANNEL** F and vice versa.

MIRRORY
The contents of all selected **SOUND CHANNELS** in the **EDIT GRID** are mirrored at the middle y-axis. **TIC** 1 will become **TIC** 32 and vice versa.

SHIFTX
The contents of all selected **SOUND CHANNELS** are moved left/right **TIC**wise. But just clicking *SHIFTX* doesn't do anything. Hold down one or two of the following keyboard buttons while you click:

with *<SHIFT LEFT>*
contents of all selected **SOUND CHANNELS** are scrolled to the left side. The leftmost TIC which is scrolled out comes back in as rightmost **TIC**.

with <*SHIFT LEFT/ALTERNATE*>
this does pretty much the same like <*SHIFT LEFT*> alone. Only all **TICs** falling out on the left side disappear. The **TICs** coming in from the right side are cleared.

with <*SHIFT RIGHT*>
refer to <*SHIFT LEFT*> and change directions.

with <*SHIFT RIGHT/ALTERNATE*>
refer to <*SHIFT LEFT/ALTERNATE*> and change directions.

SHIFTY
This again does nothing unless you do it:

with <*SHIFT RIGHT*>
Scrolls all **SOUND CHANNELs** up. The uppermost **CHANNEL** comes back in at the bottom.

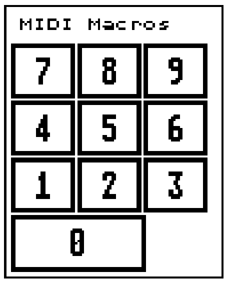
with <*SHIFT RIGHT/ALTERNATE*>
Scroll all **SOUND CHANNELs** up. But this time the **CHANNEL** scrolled out on top is gone. All **CHANNELs** coming in from the bottom are empty.

with <*SHIFT LEFT*>
You guessed right. Look at <*SHIFT RIGHT*> and change direction downwards.

with <*SHIFT LEFT/ALTERNATE*>
Refer to <*SHIFT RIGHT/ALTERNATE*> and change directions.

3.3.1.1. **MIDI Macro** Launch Pad

This little box represents the ten number keys of the ST' s numberpad. Functions are corresponding with the numberpad keys 0-9.

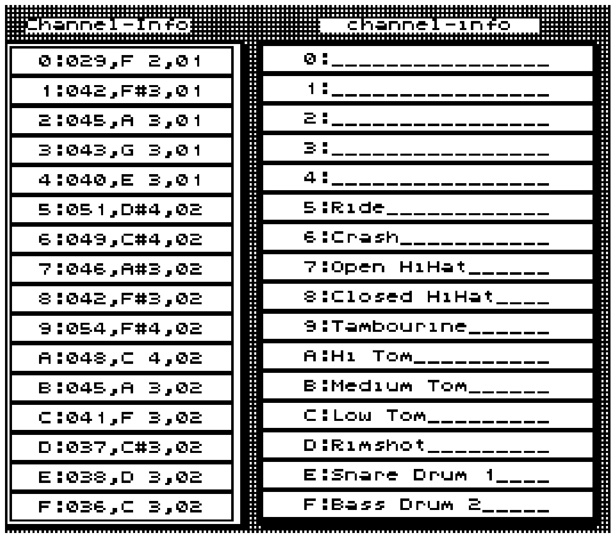


As a default each number key is assigned to the number 1 **MIDI Macro**. You can use the **MIDI Macro Key Box** to assign any available **MIDI Macro** to the number key you want [4.5.2.].

Pressing the respective number key causes PATTERNER to send the corresponding **MIDI Macro** to the MIDI Out port. This works at all times and is not affected if a playback is running or by the status of any flags. Also look at [*MIDI Macros* 2.5.].

3.3.2. Extended Channel Info

Switch to Extended Channel Info using the Graphic Commands/Extended Channel Info switch button [3.3.]. This display will come up over the **GRAPHIC COMMANDS** area:



```
Channel-Info        channel-info
0:029,F 2,01        0:________________
1:042,F#3,01        1:________________
2:045,A 3,01        2:________________
3:043,G 3,01        3:________________
4:040,E 3,01        4:________________
5:051,D#4,02        5:Ride____________
6:049,C#4,02        6:Crash___________
7:046,A#3,02        7:Open HiHat______
8:042,F#3,02        8:Closed HiHat____
9:054,F#4,02        9:Tambourine______
A:048,C 4,02        A:Hi Tom__________
B:045,A 3,02        B:Medium Tom______
C:041,F 3,02        C:Low Tom_________
D:037,C#3,02        D:Rimshot_________
E:038,D 3,02        E:Snare Drum 1____
F:036,C 3,02        F:Bass Drum 2_____
```

It displays the **SOUND CHANNEL** names you entered into the User Name fields within the **SCALE EDITOR Box** for the current **SCALE**.No G**RAPHIC COMMANDS** are available in this mode. This is exclusively used for your information and doesn' t affect playback.

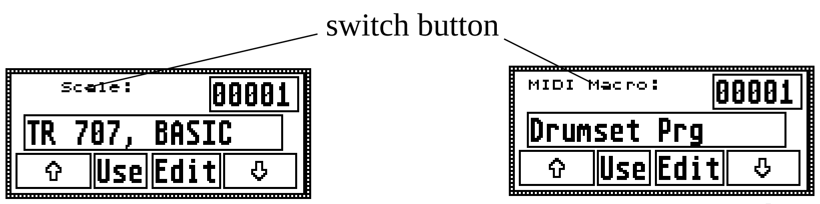
The contents of this box are changed if scroll commands are used on the current **SCALE**. Contents stay untouched after transposing.

Editing of this information is only possible in the **SCALE EDITOR Box** in Text Edit mode[*SCALES* 2.2].

Switch back to **GRAPHIC COMMANDS** by clicking on the Graphic Commands/Extended Channel Info switch button [3.3.].

3.4. SCALE/MIDI Macro Switch Box

This box operates in two modes: **SCALE** mode and **MIDI Macro** mode. Toggle between both modes by left-clicking over the switch button:



SCALE mode **MIDI Macro** mode

3.4.1. **SCALE** mode

In **SCALE** mode the switch box displays the number and name of the current **SCALE**. Use the arrow buttons to scroll up/down through the **SCALE BANK**.

EDIT
This button opens up the **SCALE EDITOR Box** [2.2.].

You can also press <*ALTERNATE E*> for this function.

USE
This button assigns the **SCALE** displayed in the switch box to the current **BEAT**.

The function can also be called by pressing <*ALTERNATE U*> on the computer keyboard.

3.4.2. **MIDI Macro** mode

In **MIDI Macro** mode number and name of the current **MIDI Macro** are displayed. Use the arrow buttons to scroll up/down through the **MIDI Macro BANK**.

EDIT
The *EDIT* button opens up the **MIDI Macro Compiler** [2.5.].

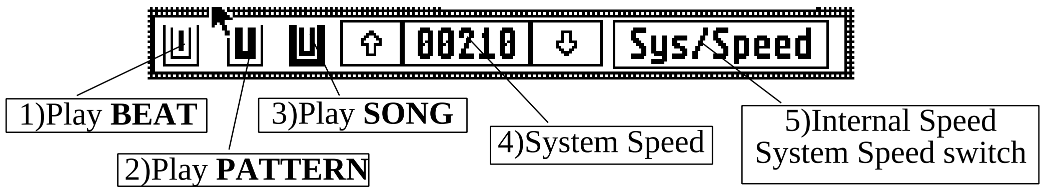
Also press <*ALTERNATE E*> to call this function.

USE
This button assigns the **MIDI Macro** in the switch box to the current **BEAT**.

This function can also be called with <*ALTERNATE U*>.

3.5. Playback menu

This small menu contains some important commands for playback. Actually the *STOP* button belongs to the same group but has been placed away from this to avoid accidental stopping while fiddling in the playback menu.



1) Play **BEAT**

Single-leftclick to start playback of current **BEAT**. The playback indication in the upper right screen corner starts flashing and the '*playing BEAT...*' info appears right to the GEM menubar. The <*B*> key on the keyboard has the same function.

Hold down the right mouse button while you click to just see the info.

2) Play **PATTERN**

Start the current **PATTERN** by left-clicking over this. Playback indication and '*playing PATTERN...*' info appears. Press <*P*> on the keyboard to launch the current **PATTERN**.

Here too, hold down the right mouse button while you click to see the info.

3) Play **SONG**

Launch current **SONG**, playback indication and '*playing SONG...*' info. This can also be started by pressing <*S*> on the keyboard.

Again, you get just the info when holding down the right mouse button in addition.

4) System Speed

SYSTEM SPEED is the speed with which the playback routine scans the **TICs** of a **BEAT**.

Set the **SYSTEM SPEED** using the up/down arrow buttons.

Speed can be changed within a range of 1-255. Note that **SPEED** is no value compatible to beats per minute on a metronome. A speed value of zero(0) stops playback (because it takes an infinity to play a **BEAT**). Avoid using **BEATs** with a **SPEED** of zero especially during playback of **CYCLEd SYSTEMs**. Theoretically a **SYSTEM SPEED** of zero is not possible - individual **BEATs**, though, could appear with their **INTERNAL SPEED** set to zero (***CYCLES*** do that sometimes).

High playback speeds (200+) can slow down other program functions (especially if the NOTE OFF FLAG is set [4.4.2.]).

5) **SYS/SPEED**

Each **BEAT** can either use the global **SYSTEM SPEED** (see at 4) or its own individual playback speed independent from **SYSTEM SPEED**.

If you left-click over this button you reverse the <*SYS*> speed flag in the current **BEAT** thus setting it to its **INTERNAL SPEED**. This is documented in the info string below the B**EAT EDIT GRID**. As a default every **BEAT'S INTERNAL SPEED** is set to 120. If the **BEAT** is already set to its **INTERNAL SPEED** it reverses to <*SYS*>.

Left-clicking while holding down the right mouse button takes over the **SYSTEM SPEED** into the current **BEAT** and sets it to **INTERNAL SPEED**.

A **BEAT** keeps its **INTERNAL SPEED** (even if it doesn' t use it) until it is changed by the above action.

Most functions within the PATTERNER including **GRAPHIC EDITING** and most **BANK**ing functions can be carried out while playing back. Playback will be interrupted, however, if any disk operation is requested. After successfully saving, playback continues.

Be careful when working in the **BANK EDITORS** or the **PATTERN/SONG EDITOR Boxes** especially if you delete or clear something (the playback routine may just be playing back the structure you are about to kill...).

3.6. UPDATE button

I am sure you have already played back some things. Did you notice that the screen is not updated automatically? There are many reasons for that (for one: how could you change the contents in the **EDIT GRID** if the **BEAT** displayed in it would never stay the same?).

Clicking over the *UPDATE* button always redraws the screen according to the current playback status - the **BEAT** that is played, the **SCALE** and **MIDI Macro** assigned to it....

In general this button is just used to get information about what the program is doing while it' s playing back. It is especially useful if you use **CYCLEs** to control playback, because here you can set up an automatic redraw (***CYCLER***). Keep in mind, though, that this function works only correct as long as playback is running.

Pressing the <*HELP*> key on the ST keyboard also carries out an *UPDATE*.

3.7. PAUSE button

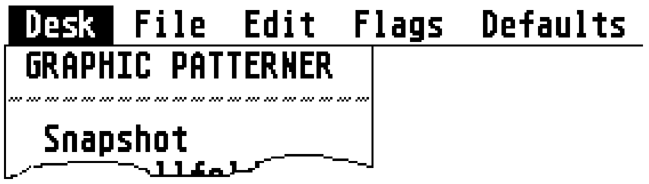
Like in a tape recorder the *PAUSE* button is used to stop playback at any point and to resume at the same spot after clicking the button again.

The *PAUSE* button is only available if the program is playing back. As long as the program is in PAUSE mode the *PAUSE* button is displayed reverse.

Stopping playback (*STOP* button/<*ESC*>) or launching a new **BEAT, PATTERN** or **SONG** resets the *PAUSE* button.

4. GEM Menubar

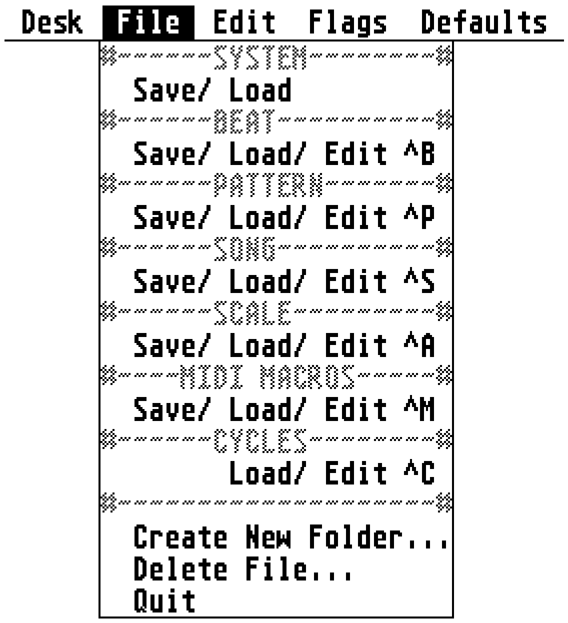
Like other GEM based programs PATTERNER uses a GEM menubar to give you access to many of its functions. Especially global load, save and edit functions are accessed via the menu. Also some important flags and system settings are available from the menubar.



4.1. DESK

4.1.1. GRAPHIC PATTERNER

This brings up the usual GEM box offering information about the program and its author. In addition this one displays information about memory allocation within the program, memory still available and your program registration number.



4.2. FILE

Under the file menu you find global save, load and edit functions for **SYSTEMs**, all the **BANKs** and also options to create new folders on disk or delete files. It also contains the program exit .

4.2.1.**SYSTEM**

SYSTEM SAVE

This option lets you save the contents of all **BANKS** currently in memory, or just a subset of them. You find details about this operation in [2.6.]

SYSTEM LOAD

This loads files which have previously been stored by using the *SAVE SYSTEM* option. Look up further information in [2.6.].

4.2.2. **BEAT BANK**

BEAT BANK SAVE

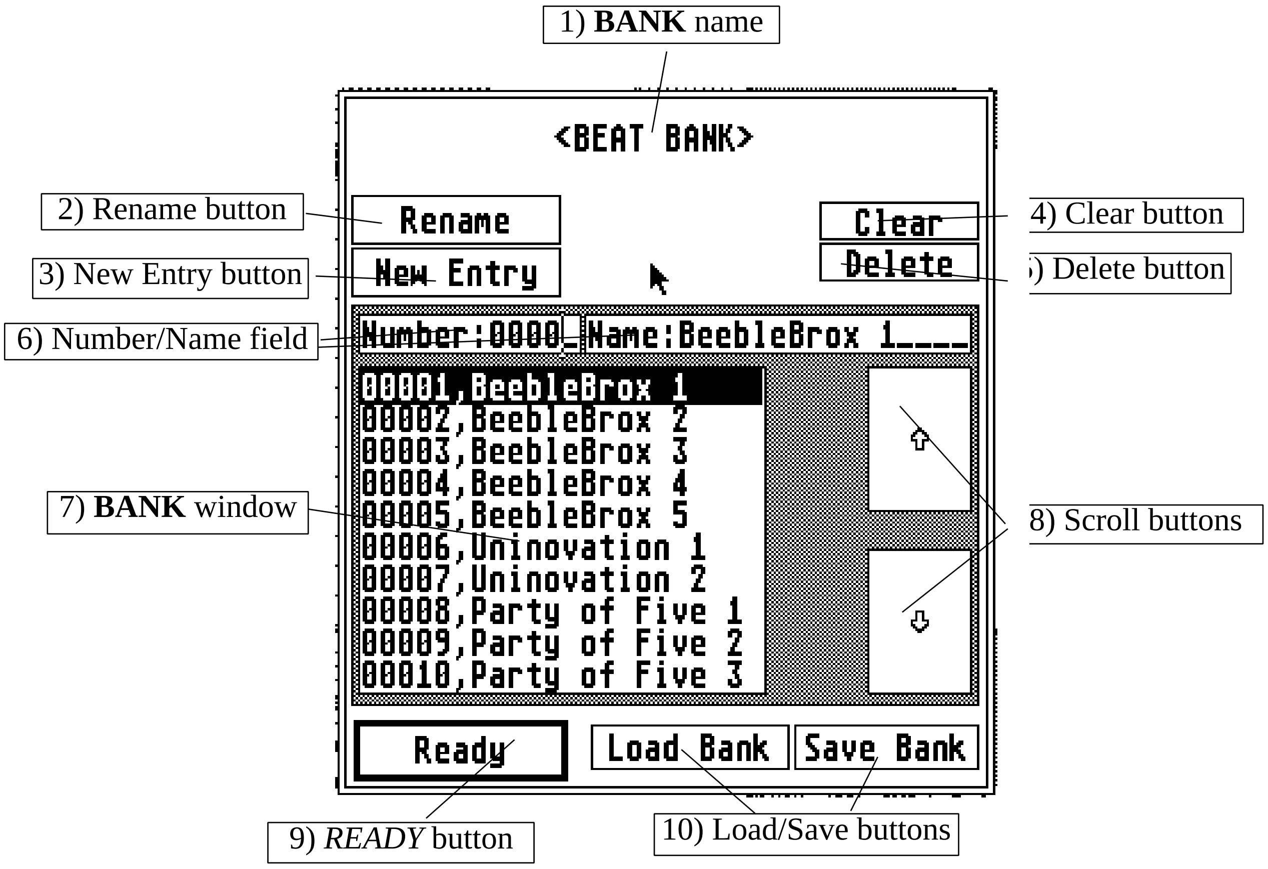
With this option you can save the **BEAT BANK** currently in memory. This uses the regular GEM Item Selector. **BEAT** files should be named with a *.BEA* extension. Select *OK* or *CANCEL* to proceed or abort the operation.

BEAT BANK LOAD

This lets you load a **BEAT BANK** from disk. Use the GEM Item Selector for your selection. Only files with the *.BEA* extension are displayed. Select *OK* or *CANCEL* to proceed or abort the operation. The **BANK** in memory is lost with this action.

BEAT BANK EDIT

This brings up the **BEAT BANK EDITOR Box** which is composed of many passive and active parts and is used to do global work in the **BEAT BANK**. You will be using an identical GEM box to do editing in all **BANKs**. So the following explanations are true for all of them.



Number:0000_ Name:BeebleBrox 1____

```
00001,BeebleBrox 1
00002,BeebleBrox 2
00003,BeebleBrox 3
00004,BeebleBrox 4
00005,BeebleBrox 5
00006,Uninovation 1
00007,Uninovation 2
00008,Party of Five 1
00009,Party of Five 2
00010,Party of Five 3
```

1) **BANK** name

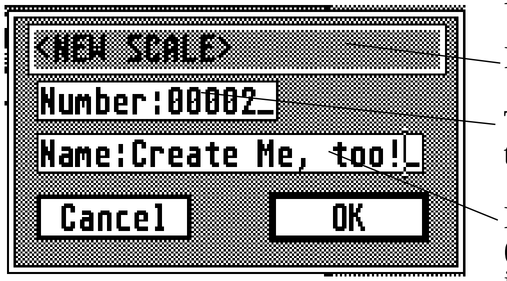
Type of **BANK** displayed in this box. This is no button and is just here for your information. As mentioned before, this very same box is used to edit all available types of **BANKs** (which are:**BEAT BANK**, **PATTERN BANK**, **SONG BANK**, **SCALE BANK** and M**IDI Macro BANK**)

2) Rename button

To rename a **BANK** entry bring the text cursor into the name field by either clicking over it or using the cursor keys on the computer keyboard. Then enter the new name into the name text field. Use the <*ESC*>, <*DELETE*> etc...keys to edit within the text field, like in any regular GEM text edit field. To take over the contents in the name field as a new entry name, click the *RENAME* button. After that the selected entry is displayed in the name and number field and is highlighted in the **BANK** window by reverse video.

3) New Entry button

This opens up the **NEW Entry Box**. If you are working in the **BEAT BANK** you will see another **NEW Entry Box** than in any of the others.



Here's the general **NEW Entry Box**:

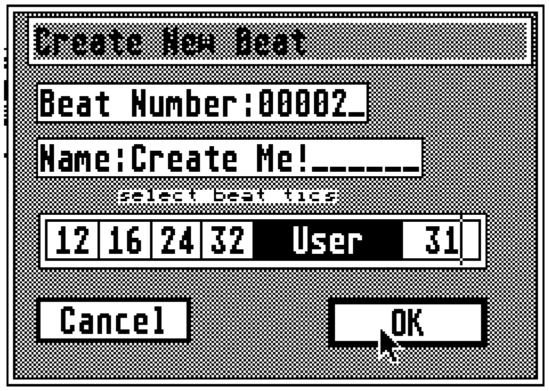
In the first line you see what you are going to create;

The number field displays the **BANK** entry number of this new entry (you can't edit this!);

Enter the name of the new entry into the name field (you don't have to do it now, you can also do that later in the **BANK EDITOR Box**);

Click the *OK* button to actually create the new entry. Or click *CANCEL* to forget about it.

This is the **NEW BEAT Entry Box**.



As you can see it contains the same buttons and text displays like the previous box.

Only in addition you can set how many **TICs** the new **BEAT** will have.

You can select one of the offered fixed values (12,16,24 or 32) or select the *USER* button and enter any value from 0-32 into the edit field right to it. You can enter values above 32 but they will be truncated to 32.

You will also use this box if you want to change the **TICs** value of a **BEAT**. To open it for an existing **BEAT** from the **MAINPAGE** click over the **TICS**: display[2.1.].

Some things you should know about creating new **BANK** entries:

a) Whenever you create a new entry in any **BANK** the new structure is appended behind the last existing **BANK** entry.

b) If you create either a new **BEAT** or **SCALE** entry the new structure will be a copy of the current **BEAT** or **SCALE** (this should be useful, because a lot of times you need relatively similar structures). However, new entries in all other **BANKS** will be bare of all data.

c) If you need more than just a few new entries you might as well use the *EDIT* funcions from the *EDIT* menu[4.3.].

4) Clear button

Use this to clear the complete **BANK** of its contents. This is not reversible and because of that a little dialog box makes sure you really want to do it. After this there will be just one entry left (named ' No name...') which is used to build a new **BANK** if desired.

5) Delete button

This deletes the selected entry from the **BANK** without question. The next entry after the deletee will take its place (and number). Deleting the last entry resets the **BANK** to its intial state (like after a Clear operation)

6) Number/Name field

The currently selected entry is displayed by its number in the number edit field and its name in the name edit field. Both fields not only display information but also act as buttons.

Bring the cursor into either of them by clicking over it or using the <*CRSR UP*> and <*CRSR DOWN*> keys. Enter a number into the number field or a name into the name field and double-click over the respective field to start a search routine which looks for the number or name you entered. If you enter a name into the name field and double-click on it, it will start searching for the nearest match in the **BANK**. The found entry will be the new current selected entry.

7) **BANK** window

Left-click over any entry within the **BANK** window to make it the new current selection.

Double left-click over any **BANK** entry to make it the current selection AND exit the **BANK EDITOR Box** (quick selecting).

8) Scroll buttons

Single left-click over one of these buttons to scroll through the current **BANK** entry by entry. Double left-click to scroll to the last/first entry.

9) Ready button

Use this button to exit the current **BANK EDITOR Box**. All changes you made are final. Also press <*RETURN*> to exit.

10) Load/Save buttons

The Load and Save buttons within the **BANK EDITOR** Box provide the same function like the *SAVE/LOAD* menubar entries for each **BANK**. After saving or loading, the program returns to the **BANK EDITOR Box**.

After exiting any of the **BANK EDITOR Boxes** (as well as after every **MAINPAGE** redraw) all **BANK** relationships are checked. If you deleted any **BANK** entry which has been used by another **BANK** - such as a **SCALE** which was assigned to a **BEAT** - this checking routine will reset those assignments which are out of range. This is displayed as an error message.

4.2.3. PATTERN BANK

PATTERN BANKSAVE

This option lets you save the current **PATTERN BANK** The regular GEM Item Selector is used. **PATTERN**files should be named with a *.PAT* extension. Select *OK* or *CANCEL* to proceed or abort the operation.

PATTERN BANKLOAD

Here you can load a **PATTERN BANK** from disk. Use the GEM Item Selector to select the **PATTERN BANK** Only files with the *.PAT* extension are displayed. Select *OK* or *CANCEL* to proceed or abort the operation. Loading a new **PATTERN BANK** will erase the one already in memory.

PATTERN BANKEDIT

This opens up the **PATTERN EDITOR Box** Since this works just the same like the **BEAT EDITOR Box** you might as well check there [4.2.2.].

<*CONTROL P*> on the computer keyboard also brings up this box.

<u>4.2.4. SONG BANK</u>

SONG BANK SAVE

This option lets you save the current **SONG BANK**. The regular GEM Item Selector is used. **SONG** files should be named with a *.SNG* extension. Select *OK* or *CANCEL* to go ahead or abort the operation.

SONG BANK LOAD

Load a **SONG BANK** from disk. Use the GEM Item Selector to select the **SONG BANK**. Only files with the *.SNG* extension are displayed. Select *OK* or *CANCEL* to proceed or abort the operation. This will erase your old **SONG BANK** in memory.

SONG BANK EDIT

This uses the same **BANK EDITOR Box** like the **BEAT BANK** [4.2.2.].

<*CONTROL S*> on the computer keyboard also brings up this box.

4.2.5. SCALE BANK

SCALE BANK SAVE

This option lets you save the curent **SCALE BANK**. The GEM Item Selector is used. S**CALE** files should be named with an *.ASS* extension. Select *OK* or *CANCEL* to proceed or abort the operation.

SCALE BANK LOAD

This lets you load a **SCALE BANK** from disk. Use the GEM Item Selector to make your selection. Only files with the *.ASS* extension are displayed. Select *OK* or *CANCEL* to proceed or abort the operation.

SCALE BANK EDIT

Refer to **BEAT BANK EDIT** [4.2.2.] for the use of the **BANK Editor Box** which is also used to work within the **SCALE BANK**.

<*CONTROL A*> on the computer keyboard also brings up this box.

4.2.6. MIDI Macro BANK

MIDI Macro BANK SAVE

With this you can save the current **MIDI Macro BANK**. The GEM Item Selector is used for this. **MIDI Macro** files should be named with a .*MMC* extension. Select *OK* or *CANCEL* to proceed or abort the operation.

MIDI Macro BANK LOAD

Here you load a **MIDI Macro BANK** from disk. Use the GEM Item Selector to select the M**IDI Macro BANK**. Only files with the .*MMC* extension are displayed. Select *OK* or *CANCEL* to proceed or abort the operation.

MIDI Macro BANK EDIT

Refer to **BEAT BANK EDIT** [4.2.2.] for the use of the **BANK Editor Box** which is also used to work within the **MIDI Macro BANK**.

<*CONTROL M*> on the computer keyboard also brings up this box.

The following two selections are both dealing with the **CYCLE** system.

4.2.7. CYCLES

LOAD **CYCLES**

This brings up the GEM Item Selector. All files with a *.CYC* extension are displayed. Only after a **CYCLER** file has been loaded successfully are the five **CYCLE** flags in *FLAGS* menu available. Whenever you load a new **BEAT, PATTERN, SONG, SCALE** or **MIDI Macro** these flags become unavailable. Thus, **LOAD CYCLES** has to be performed again.

EDIT **CYCLES**

This brings up the **CYCLE Editor Box**. Refer to the **CYCLER** section.

<*CONTROL C*> on the computer keyboard also brings up this box.

4.2.8. CREATE NEW FOLDER...

This option lets you create a new subdirectory (Folder) without leaving the program. Because of the big amount of different file types needed for the PATTERNER, it is recommended to make use of folders to avoid confusion. Each **SYSTEM** should have its own folder containing all related files.

After you clicked this selection you get the GEM Item Selector box. Select a path just as you would normally. This path will be the location for the new folder. Enter a filename in the file selection (maximum 8 characters and 3 extenders) which will be the name of your new folder. You can still *CANCEL* the operation. After clicking *OK* (or pressing <*RETURN*>) the folder is created. There may be no folders with equal names within the same path level.

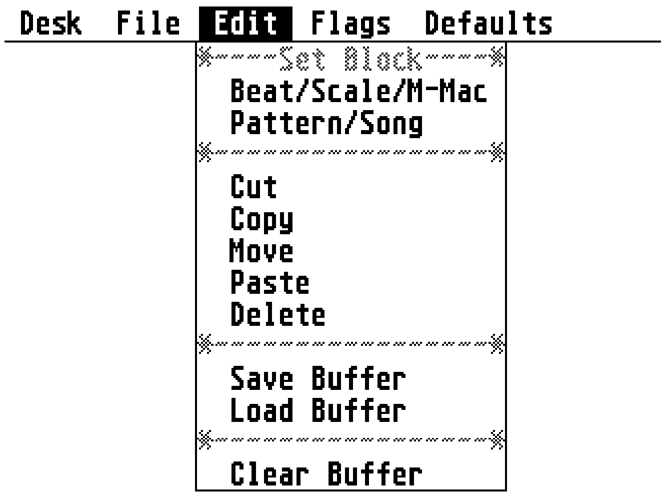
4.2.9. DELETE FILE...

Single files (of all types - but no folders) can be erased from a disk without leaving the program.

After selecting this option you are in the GEM Item Selector box. Select the file to be deleted like you would select a file to load/save (set path and filename). You can still*CANCEL* the operation. After clicking *OK* (or double-clicking your selection) the selected file is deleted (and gone forever).

4.2.10. QUIT

This does just what you expect: It' s closing down the program and returns you to the desktop after a savety request.



4.3. EDIT

All options within the *EDIT* menu are used to copy, delete and paste data within the various structures in PATTERNER.

Because this program uses a lot of different data formats these *EDIT* operations are not as easy to handle like for example in a word processor.

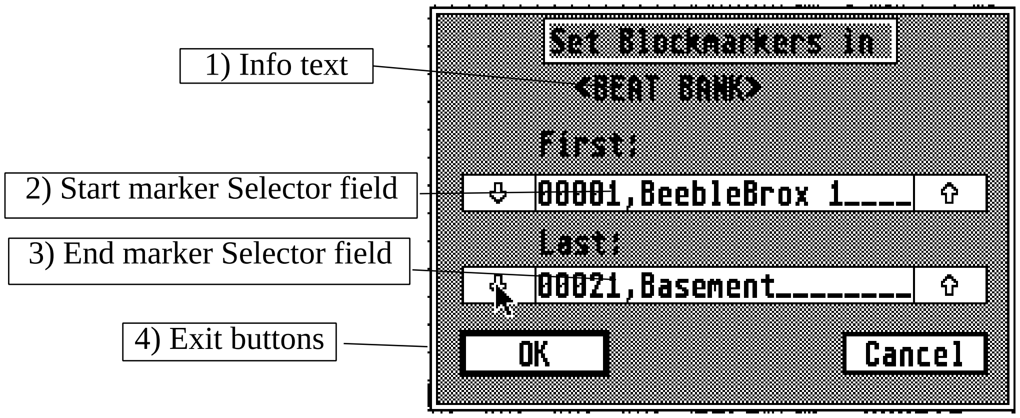
An *EDIT* function usually starts with marking a block of data within a selected **BANK**.

To mark a block you have to set its start (first) entry and end (last) entry.

You may want to know that all **MICRO COMMAND** functions [*DEFAULTS* menu, 4.5.3.] can be set to work on *SINGLE* elements (current **BEAT, SCALE**, etc), *GLOBAL* (all elements in a **BANK**) and *LOCAL* (all elements in a set **BLOCK** defined here).

4.3.1. Set Block in **BEAT BANK**

This brings up the **BLOCK MARKER Box** for the **BEAT BANK**.



1) Info text

Identification text telling you in which **BANK** you are about to set your markers. All five **BANKs** are available for *BLOCK EDIT* operations: **BEAT BANK, PATTERN BANK, SONG BANK, SCALE BANK** and **MIDI Macro BANK**.

2) Start marker Selector field

This text field displays the position of the **BLOCK** start within the current **BANK**. Use the arrow buttons to scroll up/down through the **BANK**. The marker is set right at the selected entry - thus, this selection is included in the **BLOCK**.

3) End marker Selector filed

In this text field you see the position of the **BLOCK** end within the current **BANK**. It can be set using the arrow buttons. The second marker is set right at the displayed entry, making it the LAST part of the **BLOCK**. To select just one BANK entry both marker text fields must contain the same number and name.

4) Exit buttons

Click over *OK* to confirm the **BLOCK** selection.

If the FIRST selection (Blockstart) is higher than the SECOND (Blockend) you' ll get a warning and the **BLOCK MARKER Box** pops up again. Correct the selections or:

Click *CANCEL* to forget about it.

CANCEL leaves everything like it was before calling the **BLOCK MARKER Box** (no **BLOCK** selected at all).

As soon as useable markers have been set in a **BANK**, the marked **BLOCK** is limited to the **BANK** it has been set in. You can' t *PASTE* a **SCALE BLOCK** into the **MIDI Macro BANK** or something like that.

4.3.2. Set Block in **SCALE BANK**

This lets you set the **BLOCK** markers in the **SCALE BANK**. See Set Block in **BEAT BANK**[4.3.1.] for information about setting markers.

4.3.3. Set Block in **MIDI Macro BANK**

Set **BLOCK** markers in the **MIDI Macro BANK**. See Set Block in **BEAT BANK**[4.3.1.] for more information on how to set markers.

4.3.4. Set Block in **PATTERN BANK**

Here you can set **BLOCK** markers in the **PATTERN BANK** See Set Block in B**EAT BANK**[4.3.1.] for more information about setting markers.

4.3.5. Set Block in **SONG BANK**

Set **BLOCK** markers in the **SONG BANK**. See Set Block in **BEAT BANK**[4.3.1.] for more.

The following menu options are only enabled if a proper **BLOCK** has been set in one of the **BANKs** [4.3.1. to 4.3.5.].

4.3.6. CUT

The marked **BLOCK** is copied to a buffer. It is NOT removed from the **BANK** it came from. After this operation *PASTE* and *CLEAR BUFFER* are enabled. The selected **BLOCK** also stays selected until you set a new one. If you decide to delete the **BLOCK** from it' s source **BANK** you can now select *DELETE*. This will leave only the copy of your **BLOCK** in the buffer.

4.3.7. COPY

This option makes a copy of the marked **BLOCK** by inserting all selected entries in front of the current position. A successful *COPY* operation deletes the **BLOCK** markers because the source **BANK** is re-organized during this process.

The *COPY* function can be used to create multiple **BEATs, PATTERNs, SONGs, SCALEs** or **MIDI Macros** instead of using the NEW option from the **BANK Editor Box**, which just creates one entry at a time.

4.3.8. MOVE

MOVE essentially does the same like *COPY* with the exception that the marked **BLOCK** is *DELETED* at its original position. The **BLOCK** is inserted in front of the current position. After a *MOVE* operation **BLOCK** markers are cancelled.

The affected **BANK** does not change its size.

4.3.9. DELETE

This *DELETES* the marked **BLOCK** from whatever **BANK** the markers have been set in. If all entries of a **BANK** are marked and *DELETED* the affected **BANK** is set to its DEFAULT state (1 entry). All **BLOCK** markers are removed after *DELETE*.

4.3.10. SAVE

With this you can save a filled **BUFFER** to disk. This selection is disabled if no buffer has been *CUT* or *LOAD*ed. It brings up the GEM Item Selector where you put in the path and name for the file you want to save.

Buffer files use an *.BUF* ending, regardless of their type (**BEAT, SCALE, SONG**.etc). Be sure to name *.BUF*-files in a way which enables you to recognize what **BANK** the **BLOCK** came from. Saved *.BUF* files are containing information about their source **BANK** (so you are unable to *PASTE* a **BEAT** into the **SCALE BANK**).

4.3.11. LOAD

This lets you load a *.BUF* file using the GEM Item Selector. If there is already something in the buffer it will be overwritten. Also the *PASTE*, *CLEAR* and *SAVE* options become available after a successfull *LOAD*. *LOAD*ed buffers are always *PASTE*d into their respective **BANK**.

The following two options are only enabled if a **BLOCK** has been successfully *CUT* or *LOADED* into the **BUFFER**.

4.3.12. PASTE

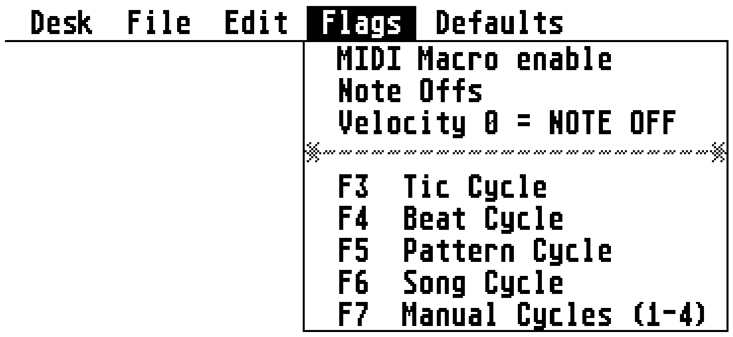
The contents of the **BLOCK BUFFER** are inserted in front of the current position into the respective **BANK**. After the **BUFFER** has been **PASTE**d its contents are still intact, so you can *PASTE* multiple times. This can be used to transfer data from one **SYSTEM** to another.

4.3.13. CLEAR BUFFER

The contents of the **BLOCK BUFFER** are cleared. All **BLOCK EDIT** options are disabled after this.

Note: **BANKs** in PATTERNER are organized in loops. If you scroll past the last entry you come back to number one. So, theoretically, whenever you wanted to *COPY, PASTE* or *MOVE* something to the very end ('behind' the last entry) of a **BANK** you'd have to use entry number one(1) as insertion point - but this will put the **BLOCK** right in front of everything instead at the very end. An easy solution is to create a DUMMY (unused) entry. Keep it the last entry (by adding **BANK** entries using **BLOCK EDIT** commands exclusively) and set the insertion pointer right on it before *COPY*, *PASTE* or *MOVE* and your data will be moved to the **BANK** end - in front of the DUMMY. On the other hand it doesn't matter in what order the entries in a **BANK** are - except for the reason of keeping them in a special order.

Keep in mind, though, that **BLOCK EDIT**ing involves a lot of re-organization within the affected **BANKs**. Especially if entries are switching positions within a **BANK** the structure of an existing **SYSTEM** will be disrupted. Whenever **BANK** entries are *DELETED* PATTERNER will tell you if the SYSTEM structure has been changed.



4.4. FLAGS

Under the *FLAGS* menu you find several toggle switches which alter playback and other functions of the program. If the respective menu entry is selected it receives a check mark or the check mark is cleared and the flag with it. Here' s a closer look at all available flags.

4.4.1. **Enable MIDI Macro Flag**

As we learned in **MIDI Macros** [2.5.] every **BEAT** is assigned to a **MIDI Macro** which can be sent out through the MIDI Out port just before the respective **BEAT** is played. This flag lets you switch this function on or off on a global level (i.e. for every **BEAT**).

The status of each **BEAT** s individua**MMac Flag** is not affected. However, **MIDI Macros** are enabled only if BOTH, **Enable MIDI Macro Flag** and the individual **BEAT MMac Flag** of the current **BEAT** are ticked [2.1.].

The status of this flag is saved along with the **SYSTEM**.

4.4.2. **Note Offs flag**

Take a look at the **BEAT EDIT GRID**[2.1.]. Usually you see a couple of filled **GRID POSITIONs** while all others are clear. The **NOTE Offs Flag** determines how clear **GRID POSITIONs** are interpreted during playback. If this flag is clear, only the NOTE On commands of filled **GRID POSITIONs** are sent, clear **POSITIONs** are ignored. This usually lets the synthesizer sounds go on until they fade by themselves, the same key number is called again or the keyboard runs out of voices. Is the **NOTE Off Flag** ticked the PATTERNER sends a NOTE Off for every clear **GRID POSITION** which creates something like a staccato effect (depending upon the sound you use).

The use of this flag usually is determined by the sounds of the MIDI device you use. If this flag is set ALL notes are silenced immediately after they have been played (to be exact, if the next **TIC** on the same **SOUND CHANNEL** is clear). Also playback speed becomes slower when this flag is set because more data is processed.

If you want to have individual control over when particular notes are terminated with a NOTE Off you will want to make use of this next flag.

4.4.3. VELOCITY 0 = NOTE OFF

With this flag you can control the function of NOTE On entries with a velocity of zero(0) in all **BEATs**. Is this flag cleared PATTERNER sends out NOTE On commands with velocity zero(0) just as displayed in the **GRID**. In most cases a velocity sensitive keyboard (or sound generator) will interpret velocity zero like a NOTE Off message.

However, if the flag is set, all NOTE On commands with velocity zero are actually sent as NOTE Off commands.

CYCLE Control Flags

The following four flags are only enabled after a **CYCLE** file (using a *.CYC* suffix) has been loaded successfully. In addition they are disabled after any other loading operation. This is necessary because the **CYCLE** file is relocated after loading in relation to ALL **BANKs** in memory.

Each **CYCLE** Flag in selected position adds individual program parts to the PATTERNER playback routine. The added parts are user defined and assembled by the **CYCLER** section.

For more about **CYCLEs** read the **CYCLER** section of this book.

4.4.4. **TIC CYCLE**

When this flag is on the **TIC CYCLE** subroutine is used after every played **TIC**. If cleared the subroutine is not used. Refer to **CYCLER** section for further information.

Pressing the <*F*3> key on the ST keyboard has the same function.

4.4.5. **BEAT CYCLE**

If this flag is set the **BEAT CYCLE** subroutine is used after every played **BEAT**.

This is the same function like pressing <*F4*> on the ST keyboard.

4.4.6. PATTERN CYCLE

Is this flag set the **PATTERN CYCLE** subroutine is used after every played **PATTERN**

Pressing <*F5*> on the ST keyboard does the same.

4.4.7. **SONG CYCLE**

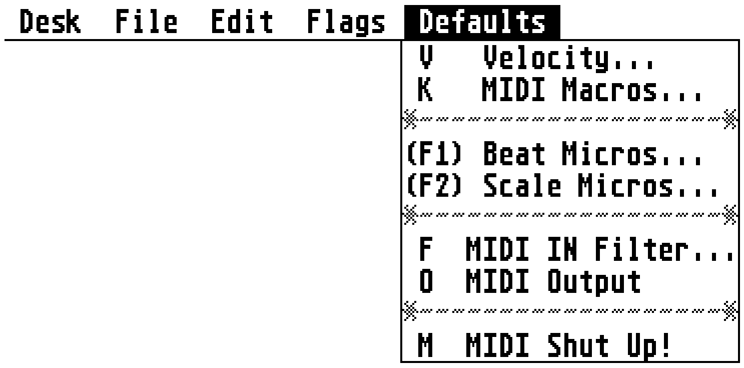
If this flag is set the **SONG CYCLE** subroutine is used after every played **SONG**.

This flag can also be inverted by using the <*F6*> key on the ST keyboard.

4.4.8. **MANUAL CYCLE**

If this flag is set you can use all four manual subroutines using the keyboard keys *<1>* to *<4>*. Further information in the CYCLER section of this book.

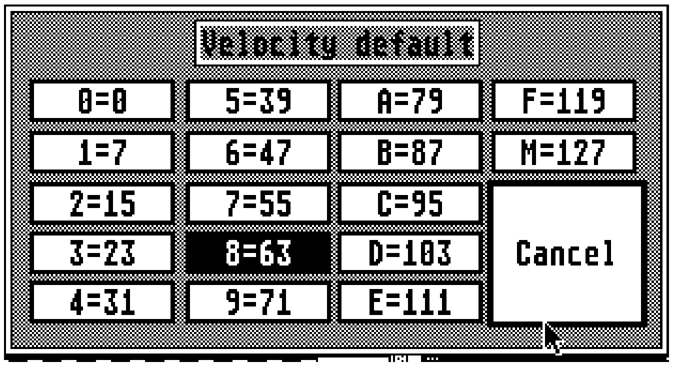
Pressing the *<F7>* key on the ST keyboard has the same function.


Desk File Edit Flags Defaults
V Velocity...
K MIDI Macros...
(F1) Beat Micros...
(F2) Scale Micros...
F MIDI IN Filter...
O MIDI Output
M MIDI Shut Up!

4.5. DEFAULTS

4.5.1. VELOCITY..

This GEM box lets you set the DEFAULT VELOCITY. You can also call the VELOCITY Box by pushing the <V> key on the ST keyboard.


Velocity default

0=0	5=39	A=79	F=119
1=7	6=47	B=87	M=127
2=15	7=55	C=95	Cancel
3=23	8=63	D=103	
4=31	9=71	E=111	

The value you select in this box is used when:

a) The current **BEAT** is being **FILL**ed or manipulated by other graphic commands.

b) A **TIC** is filled clicking upon it.

c) **DEFAULT VELOCITY** is also used by many **CYCLE** subroutines.

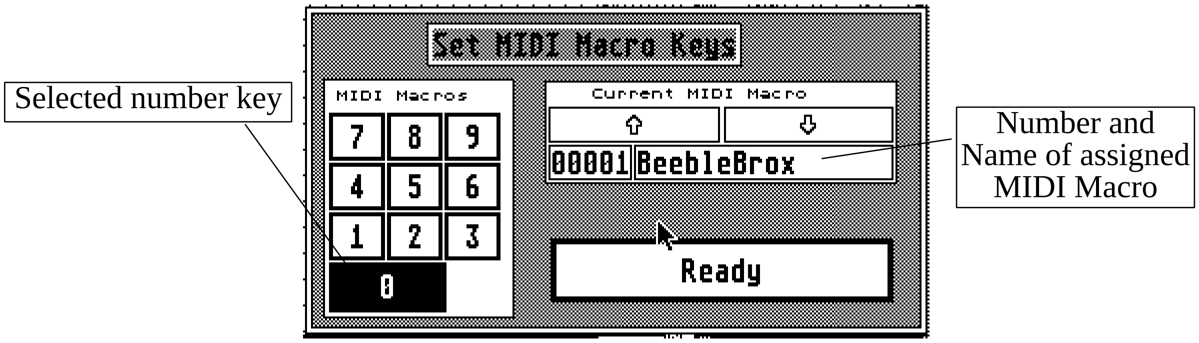
Regular MIDI velocity is within a range from 0-127 (minimum to maximum). Because of the one digit display limitation in the **BEAT GRID** only 16 different velocity levels are available. They are displayed in symbols from zero(0) to 9, then A to F and M for maximum in the mono-chrome version. The color version of PATTERNER uses three colors (black, green, red) and five different fill states. Here a conversion list:

Value	displayed	Interpretation	
Mono	Color	Decimal	Hexdecimal
0	crossed	0	$0
1	black 2	7	$7
2	green 2	15	$F
3	red 2	23	$17
4	black 3	31	$1F
5	green 3	39	$27
6	red 3	47	$2F
7	black 4	55	$37
8	green 4	63	$3F
9	red 4	71	$47
A	black 5	79	$4F
B	green 5	87	$57
C	red 5	95	$5F
D	black 6	103	$67
E	green 6	111	$6F
F	red 6	119	$77
M	red/green/black	127	$7F

All buttons in the **VELOCITY Box** are exit buttons. If you exit the **VELOCITY Box** with *CANCEL* **DEFAULT VELOCITY** will stay like it was. Any other button sets the D**EFAULT VELOCITY** value to the number shown on the respective button.

4.5.2. MIDI Macros...

This brings up the **MIDI Macro Key Box**. You can also access this box if you press <*K*> on the ST keyboard from the **MAINPAGE**.



In general **MIDI Macros** can be launched from two different triggers:

1) During playback at the beginning of a **BEAT** it has been assigned to by automatic control [3.4.2.].

2) At manual control of the user by clicking over the number keys of the **MIDI Macro** launchpad in the **GRAPHIC COMMANDS** area [3.3.1.1.] or by pressing one of the number keys on the ST's numberpad.

Both trigger sources are independent from each other and can use any **MIDI Macro** contained in the **MIDI Macro BANK**.

The **MIDI Macro Key Box** is used to assign a selection of all available **MIDI Macros** to the ten available number buttons on the number pad.

If the box opens for the first time you will see the number '0' key selected and the first **MIDI Macro BANK** entry assigned to it. This setup simply means that whenever you push number key '0' on the ST number pad (or its respective button on the **MAINPAGE**) you send the contents of the assigned **MIDI Macro** to the MIDI Out port.

To assign another **MIDI Macro** to the '0' key simply use the scroll up/down arrows to select from the **MIDI Macro BANK**. Whichever **MIDI Macro** is displayed will be used with the highlighted number key.

To assign **MIDI Macros** to the other number keys click over the desired number pad button. This will be drawn reverse and the **MIDI Macro** it is connected to at the moment will be displayed in the text field. Again, use the arrow buttons to select the needed **MIDI Macro** from the **BANK**.

Use the <*READY*> button to exit this box.

The numberpad/**MIDI Macro** assignments you define in this box are kept in memory until you change it, the complete **MIDI Macro BANK** is cleared, or entries from it which are assigned to keys are deleted (in the latter two cases either the complete list, or the affected list entry will be reset to **MIDI Macro** 0001).

MIDI Macros can only be launched using the number pad or the launch pad from the **MAINPAGE**.

About **MICRO COMMANDS**

MICRO COMMANDS are subroutines which perform one specific task in one specific data structure. In default setting a **MICRO COMMAND** is limited to work in the currently selected structure, like the current **BEAT** or current **SCALE**, although work regions can be set.

Press the **BEAT MICRO COMMAND** Button <*F1*> from the **MAINPAGE** and the active **BEAT MICRO COMMAND** is performed within the current **BEAT**. Also **MICRO COMMANDS** provide the core for many edit routines.

MICRO COMMANDS can be adjusted to affect either just the current structure or all structures defined within an **EDIT BLOCK** [4.3.1.] or all structures within a complete **BANK**.

To set the **MICRO COMMAND** work area use one of the following three buttons present in every **MICRO COMMAND Box**:

SINGLE

This is the default setting. **MICRO COMMAND** calls are limited to the current **BEAT** or **SCALE**.

LOCAL

This limits the **MICRO COMMAND** work area to the selected **EDIT BLOCK**. If no **BLOCK** has been defined or the **BLOCK** is not within the right **BANK** you will see an error message.

GLOBAL

All entries in the respective **BANK** are affected.

In addition to the three mentioned options you can enable the

SCREEN REDRAW

If this button is not selected the changes in the **BANK** entries are done without showing on screen. When selected every single entry is displayed after being altered. Especially if a **MICRO COMMAND** is used on large **EDIT BLOCK**s or in **GLOBAL** mode this may take some time (and the process can' t be stopped - you have to wait!).

MICRO COMMANDS are set within the following boxes:

4.5.3. **BEAT MICRO COMMANDS...**

The active **BEAT MICRO COMMAND** is executed by pressing the <*F1*> key. Each one affects either the current **BEAT** (*SINGLE*), the set **BEAT BLOCK** (*LOCAL*) or all entries in the **BEAT BANK** (*GLOBAL*)[see: About **MICRO COMMANDS**].

New Velocity

All NOTE On commands on selected **SOUND CHANNELs** in the current **BEAT** are set to **DEFAULT VELOCITY** [see **VELOCITY**, *DEFAULTS* menu 4.5.1.].

Up Velocity

This increases the velocity of NOTE On commands as the **SOUND CHANNEL** numbers increase. The lowest **VELOCITIES** are at the top while the highest are at the bottom. Only selected **SOUND CHANNELs** are affected.

Down Velocity

This decreases the velocity of NOTE Off commands as the **SOUND CHANNEL** numbers increase. The highest **VELOCITIES** are at the top while the lowest are at the bottom. Only selected **SOUND CHANNELs** are affected.

Left Velocity

This creates a crescendo from left to right. The lowest **VELOCITIES** are at the left while the highest are at the right side. Only selected **SOUND CHANNELs** are affected.

Right Velocity

This creates a decrescendo from left to right. The highest **VELOCITIES** are at the left while the lowest are at the right side. Only selected **SOUND CHANNELs** are affected.

Random Fill

All selected **SOUND CHANNELs** are filled with a random pattern. See **MAINPAGE** about **CHANNEL SELECTION**. This usually leaves the **BEAT** very crowdy.

Random And

All selected **SOUND CHANNELs** are ANDed with a random number. This can be used to thin out data.

Random Tics

TICs of the affected **BEAT**(s) are set to a random value between 0 and 32.

Random Scale

The affected **BEAT**(s) is assigned to a randomly chosen **SCALE**. The random number is within the range of the available **SCALE BANK**.

Random MMac

The affected **BEAT**(s) is assigned to a randomly chosen **MIDI Macro**. The random number is within the range of the available **MIDI Macro BANK**.

Use Scale

The **SCALE** assignment of the current **BEAT** is set to the **SCALE** assignment of the currently assigned **SCALE**. This sounds funny (if not incomprehensible) and has no use if only one **SINGLE BEAT** is affected by the command. It helps a lot if you would want to assign all available **BEATs** or just a **BLOCK**ed subset to the same **SCALE** the current **BEAT** is assigned to.

Use MMac

The **MIDI Macro** assignment of the current **BEAT** is set to the **MIDI Macro** assignment of the current **BEAT**. Check **USE SCALE** command on further information.

SYS/Speed

The **SYS/SPEED Flag** within the affected **BEAT** is changed. A **BEAT** set on **INTERNAL SPEED** changes to **SYSTEM SPEED** (<*SYS*>). A **BEAT** set on **SYSTEM SPEED** changes to its **INTERNAL SPEED**. This changes only the flag but not the **INTERNAL SPEED** value itself.

Set Tics

The **TICs** value of all affected **BEATs** are set according to the **TICs** value of the current **BEAT**.

Flip MMC Flag

This sets/clears the individual **BEAT MIDI Macro Flag** in the affected **BEAT**(s).

4.5.4. SCALE MICRO COMMANDS...

The active **SCALE MICRO COMMAND** is launched pressing the <*F2*> key on the keyboard from the **MAINPAGE**. Each of the below commands affects either the current **SCALE** (*SINGLE*), the set **SCALE BLOCK** (*LOCAL*) or ALL entries in the **SCALE BANK** (*GLOBAL*).

Transpose up

All selected **SOUND CHANNELs** in the affected **SCALE** are tranposed up one half step. If a pitch gets out of range it is reset to ' 0' (C 0). (see **MAINPAGE** for information about **SOUND CHANNEL SELECTION** [3.2.])

*Transpose down*s

All selected **SOUND CHANNELs** in the affected **SCALE** are transposed down one half step. Pitches below minimum pitch (C 0) are set to maximum pitch.

Scroll up

The contents of all sixteen **SCALE CHANNELs** (including text information) are scrolled up one position. Data scrolled out on top is inserted at the bottom. This affects all S**OUND CHANNELs**. Command corresponds with the Scroll Up option in [2.2.].

Scroll down

The contents of all sixteen **SOUND CHANNELs** are scrolled down one position. Data scrolled out at the bottom is inserted on top. This affects all **SOUND CHANNELs**.

Reset Keys

The pitches of all selected **SOUND CHANNELs** are reset to pitch ' 0' (C 0).

Reset MIDI

The MIDI Channels of all selected **SOUND CHANNELs** are reset to MIDI Channel 1.

Set Keys

The key numbers of all affected **SCALEs** are set according to the key numbers of the current **SCALE**.

Set MIDI

The MIDI Channels of all affected **SCALEs** are set according to the MIDI Channels in the current **SCALE**.

Octave up

All selected **SOUND CHANNELs** are transposed up one octave. Check [2.2.] *Octave Up*, also. No complete octave when going up from maximum octave (' :').

Octave down

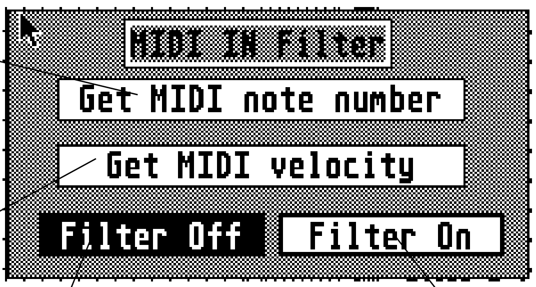
All selected **SOUND CHANNELs** are transposed down one octave. No complete octave when going down from minimum octave (' 0').

4.5.5. MIDI IN Filter...

While the PATTERNER is playing back it also stores all incoming MIDI NOTE On/Off messages in an internal buffer. The **MIDI IN Filter Box** gives you some control over which parts of the incoming messages are saved and which are deleted. This MIDI In buffer, however, can only be accessed by routines from the **CYCLER** (see **CYCLER** section).

If this button is selected, the note number of the incoming message is stored. Is the button off, the note number is filtered out.

If this one' s selected, the velocity of the incoming message is stored. Is the button off, the velocity is filtered out.



FILTER OFF
Click over this button to leave the **MIDI Filter Box** and to switch the filter off.

FILTER ON
If you leave the M**IDI Filter Box** with this button the filter will be on.

If none of the two above buttons is activated, nothing is coming in, because all three parts of the message are filtered out (status bytes are not stored in the buffer). Also, only NOTE On/Off messages are scanned and treated equal. All 16 MIDI channels are monitored (OMNI MODE).

This box can also be opened using the <*F*> button on the keyboard.

4.5.6. MIDI/Int. Output

When PATTERNER is powered up all output data is routed to the MIDI Out port of the ST. However, by selecting this option from the menubar all output can be directed to the ST' s internal sound chip. Click again to switch back to the MIDI Out port.

You will probably not use this option too often, since the internal sound chip limits you to only three independent sound channels. It comes in helpful if no keyboard is around and you want to test some PATTERNER files. I also found it useful for other purposes. However, because of several limitations of this output option you need MIDI sound sources to use the PATTERNER to its full potential.

Some details:

1) The current version of the internal output module doesn' t support any sound variations.

2) Dynamics are interpreted but translated in very rough steps. The sound chip just knows sixteen available volume levels.

3) Unlike most synthesizers the sound chip waits for a NOTE Off command to shut a note off (well, if a Note On is received while all three channels are already in use the channel which has been in use for the longest time will accept the new NOTE On). This will force you in many cases to switch on the *NOTE OFF Flag* [4.4.2.] to get any usable results.

4) The current version of the internal sound module doesn' t support all 128 MIDI note numbers. In fact, while the low notes are OK the highest octaves (96-127) had to be doubled (the sound chip is sort of unexact above 10kHz).

5) The internal sound module accepts all MIDI data. However, due to its limitations only the following MIDI messages are transformed into action:

NOTE On messages ($80)	the corresponding note is switched on
NOTE Off messages ($90)	the corresponding note is switched off
System Reset ($F0)	all notes off
All Notes Off ($B0)	all notes off

You don' t hear the usual keyclick in PATTERNER. It has been shut off to avoid conflicts with the internal sound module. (If you DO need it you have to use the Control Panel Accessory which is on your ST' s System Disk).

You can also use the <*O*> button on the ST to toggle between MIDI and Internal Output.

4.5.7. MIDI Shut Up!

Use this option to send a MIDI System RESET message and an ALL NOTES OFF command. This should terminate all stuck notes and other MIDI problems. Some MIDI devices, though, may not recognize this message.

This option can also be launched if you press the <*M*> button on the keyboard.

If you use this option while playing back you should experience no problems. But no guarantee for this! This can only be used from the **MAINPAGE**.

5. TIPS AND HINTS

5.1. The SCALE Editor Box seems to be stuck

Sometimes (not very often) - usually after a playback - the **SCALE Editor Box** doesn' t seem to work. If this occurs the program is hooked up in a MIDI request loop and is waiting for more MIDI data from the MIDI In port. The only way to get out of this loop is to press one or more keys on the connected keyboard. After this everything should work like usual.

5.2. Use the USER NAMES effectively

If you create a **SCALE** using the **SCALE Editor Box** better make extended use of the User Names. For example setting up a drumset **SCALE** is much easier to do if you enter the instrument names. It also helps you identify the instruments while entering patterns in the **BEAT GRID EDITOR**. Note, though, that *SCROLL*ing of a **SCALE** in either direction takes the info strings with their pitch values and MIDI Channels but *TRANSPOS*ing doesn' t change this string.

5.3. Use all names effeciently

In the PATTERNER you can name almost everything using a maximum of sixteen characters. It is true that this uses a lot of memory but it also makes it much easier to identify structures and list entries. Sixteen characters leave you a lot of space to put in enough information for all uses. Always try to use unique names which let you know in a second with what you are dealing. This will also make it much easier to work with the search function in the different **BANKs** and **EDITOR** Boxes.

5.4. Use the Velocity 0 value as NOTE Off substitute

Depending on the MIDI devices you use you may want to send NOTE Off messages to one or more of them during playback. There are two ways:

1) If you want EACH NOTE On to be followed by the corresponding NOTE Off you can simply set the *Note Off Flag* [4.4.2.]. However, this will cut the note (or NOTE On messages) really short. You get kind of a staccato effect. In addition, if this flag is set, sound output IS slowed down because more messages are sent.

2) Another solution can be used on velocity sensitive sound sources. Note that many synthesizers don' t provide you with a touch sensitive keyboard but still react to velocity data. With such MIDI devices you can use a Note On with velocity zero (0) in the **EDIT GRID** to cut the note off at the moment YOU desire. It is also possible to convert NOTE On messages with velocity ' 0' into real NOTE Off commands using the *Velocity 0 = Note Off flag* [4.4.3.].

6. DATA FORMATS

Data representation is different on disk than in computer memory.

Every saved file receives a file type ID header composed of a ' .' and the three letters of its file name extension. Filetypes are not interchangeable. You will get an error message if you try to load the wrong filetype.

6.1. BEAT BANK Data Format

BEAT structure format:

16	bytes - name string
2	bytes - ID number of assigned **SCALE**
2	bytes - **TICs** used within the **BEAT**
1	byte - SysSpeed flag: bit 7, **MIDI Macro** enable flag: bit 6
1	byte - SPEED value
2	bytes - ID number of assigned **MIDI Macro**
64	bytes - **BEAT** playback data (1 longword per channel)
512	bytes - **VELOCITY** data (32 bytes per channel)

BEAT BANKs are saved like this:

4 bytes - file type ID, '*BEA*'
2 bytes - number of BEAT structures within this BANK (=n)
n*600 bytes - **BEAT** structures as defined above
(next **BEAT** structure)

6.2. SCALE BANK Data Format

SCALE structure format:

16	bytes - name string
16	bytes - MIDI Keynumbers for sixteen channels
16	bytes - MIDI Channels for sixteen channels
256	bytes - 16 character User Name strings for sixteen channels

SCALE BANKs are saved ike this:

4 bytes - file type ID, '*ASS*'
2 bytes - number of **SCALE** structures within this **BANK** (=n)
n*304 bytes - **SCALE** structure as defined above
(next **SCALE** structure...)

6.3. PATTERN BANK Data Format

PATTERN structure format:

16 bytes - name string
2 bytes - number of entries in this **PATTERN**
4 bytes - pointer to Number List containing IDs of the **BEATs** used in the **PATTERN**

Number List format:

2 bytes - number of first **BEAT** to play
2 bytes - number of second **BEAT** to play
.
.

PATTERN BANKs are saved like this:

4 bytes - file type ID, '*PAT*'
2 bytes - number of **PATTERN** structures in this **BANK**
22 bytes - **PATTERN** structure as described above
n bytes - number list **PATTERN** 1
....next **PATTERN**..

6.4. SONG BANK Data Format

SONG structure format:

16	bytes - name string
2	bytes - number of entries in this **SONG**
4	bytes - pointer to Number List containing IDs of the **PATTERNs** used in this **SONG**

Number List format:

2	bytes - number of first **PATTERN** to play
2	bytes - number of second **PATTERN** to play
.	
.	

SONG BANKs are saved like this:

4	bytes - file type ID, '*SNG*'
2	bytes - number of SONG structures in this **BANK**
22	bytes - **SONG** structure as described above
n	bytes - number list

....next **SONG**...

6.5. MIDI Macro BANK Data Format

MIDI Macro structure format:

16	bytes - name string
2	bytes - number of bytes in this **MIDI Macro**
4	bytes - pointer to data field containing MIDI data

MIDI Macro data field structure:

n	bytes - data bytes terminated by $ff.

MIDI Macro BANKs are saved like this:

4	bytes - file type ID, '*MMC*'
2	bytes - **MIDI Macro** structures within this **BANK**
n	bytes - **MIDI Macro** structures as defined above

6.6. System File Data Format

SYSTEM files are built like this:

12 bytes - file name of **BEAT BANK**
12 bytes - file name of **PATTERN BANK**
12 bytes - file name of **SONG BANK**
12 bytes - file name of **SCALE BANK**
12 bytes - file name of **MIDI Macro BANK**
6 bytes - speed values
4 bytes - flags
2 bytes - name field selection flags (bits 0-4)

SYSTEM files are saved like shown above with an additional 4 byte header:

4 bytes - file type ID, '*SYS*'

<u>6.7. CYCLE File Data Format</u>

CYCLE .*SRC* files are saved like this:

4	bytes - file type ID, '*SRC*'
4	bytes - filesize longword
4	bytes: 0, *INIT* ID
4	bytes: 1, *TIC CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID
4	bytes: 2, *BEAT CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID
4	bytes: 3, *PATTERN CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID
4	bytes: 4, *SONG CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID
4	bytes: 5, *MANUAL 1 CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID
4	bytes: 6, *MANUAL 2 CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID
4	bytes: 7, *MANUAL 3 CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID
4	bytes: 8, *MANUAL 4 CYCLE* ID
n	bytes: ID' s of inserted routines
4	bytes: 9, *CYCLE END* ID

ID describes the relative position of the **CYCLE** routine in the used **CYCLE** library.

6.8. CYCLE program File Format

4	bytes - file type ID, ' *CYC*'
4	bytes - file size in bytes (n)
n	bytes - 68000 assembly code

6.9. CYCLE.LIB File Format

4	bytes - file type ID, ' *LIB*'
4	bytes - file size in bytes (n)
n	bytes - 68000 assembly code. Format is described in the **CYCLER** section

6.10. Block Buffer File Format

4	bytes - file type ID, ' *BUF*'
2	bytes - number of entries in buffer
1	byte - buffer type flag (1=beat,2=scale,3=midi macro,4=pattern,5=song)

6.11. PATTERNER internal playback access array

This array holds among others the following data which is used and updated by the playback routine. If you have to know which **TIC, BEAT** or **SONG** is being played at the moment, or want to request the **SYSTEM SPEED** - get the *WORK_LIST* base address (*WORK_LIST*(a6) in any **CYCLE** routine) and use the needed offset from the playback offsets EQU list (**CYCLER** library source code). All positions can be changed by **CYCLER** routines. Some variables, though, are declared as read-only which means, they can cause strange program behavior if changed. Also be sure to keep all values within their proper boundaries.

This array can only be accessed from **CYCLE** routines! All offsets and values are defined in the **CYCLER** library.

The following example shows you how to get a hold of the address of the Internal Sound Module:

```
move.l WORK_LIST(a6),a0
move.l INTERNAL_PTR(a0),a0
```

Offset name	Offset/Size
TIC_COUNTER	0/word

Number of **TIC** currently played (from 0-31)

PLAY_FLAG	2/word

flag; if this is <> 0 then playback is in progress.

VELOCITY0_FLAG	3/byte

flag; if this is <> 0, all Note On commands with a velocity of 0 are sent as Note Off commands.

MMAC_FLAG	4/byte

flag; if this value <> 0 then the global **MIDI Macro** change is ON (global enable **MIDI Macro flag** checked). Internal **BEAT MIDI Macro** change flag has to be checked in addition to enable the **BEAT** to send **MIDI Macros**. If flag = 0 then **MIDI Macros** are globally disabled. Read-only!

NOTE_OFF_FLAG	5/byte

global NOTE OFF flag; if this flag <>0 then every **TIC** on any **SOUND CHANNEL** which does not contain a Note On message is interpreted as Note Off. This creates a staccato effect. Read-only.

TICC_FLAG	6/byte

tic cycle flag; if this flag <> 0 then the **TIC CYCLE** is used during playback after each completed **TIC**. If flag = 0 then **TIC CYCLE** is bypassed. Read-only!

BEATC_FLAG 7/byte
beat cycle flag; flag <> 0 then **BEAT CYCLE** is used during playback after every completed **BEAT**. Bypassed if flag = 0. Read-only!

PATTERNC_FLAG 8/byte
pattern cycle flag; flag <> 0 then **PATTERN CYCLE** is used during playback after every completed **PATTERN** Bypassed if flag = 0. Read-only!

SONGC_FLAG 9/byte
song cycle flag; flag <> 0 then **SONG CYCLE** is used during playback after every completed **SONG**. Bypassed if flag = 0. Read-only!

MANUALC_FLAG 10/byte
manual cycle flag; flag <> 0 then all four **MANUAL CYCLES** are active. Launch **MANUAL CYCLES** during playback pushing the <*1*> to <*4*> keys on the ST keyboard. If flag = 0 this option is bypassed. Read-only!

VELOCITY_DEFAULT 11/byte
velocity default (0-127 in 16 steps) [4.5.1.].

SPEED 18/word
system speed value (1-255); this value contains the same value like the speed display on the **MAINPAGE**. If you change this the new value will be displayed on the **MAINPAGE** at the next screen update. To actually alter SYS SPEED you also have to change *WORK_SPEED* and *WWORK_SPEED*!

WORK_SPEED 20/word
work speed 1. If you change the *SPEED* value also adjust *WORK_SPEED* like that: *WORK_SPEED = MINUTE/SPEED* (which creates a reverse brake value). *MINUTE* is defined in the **CYCLER** library.

WWORK_SPEED 22/word
work speed 2; must be identical to *WORK_SPEED*. For example of setting/reading SYS SPEED look up the library source code.

PATTERN_BASE 24/long
PATTERN BANK base address; contains the base address of the **PATTERN BANK** Read-only!

SONG_BASE 28/long
SONG BANK base address; contains the base address of the **SONG BANK**. Read-only!

WORK_BEAT 32/word
index of **BEAT** played; this value can be used to find out which **BEAT** is currently played. You can also change the current **BEAT**, but be careful not to request **BEAT** indexes higher than are actually present in the **BEAT BANK** (find out about this using the **COUNTER** offset in a **BEAT BLOCK**).

WORK_SCALE 34/word
index of used **SCALE**; usually this contains the index of the **SCALE** of the current **BEAT**. When changing this value be careful for the same reasons mentioned in *WORK_BEAT.*

WORK_MMAC 36/word
index of used **MIDI Macro**; same like *WORK_SCALE*.

MANUAL1_CYCLE_J 58/long
address pointer; lets you use the **MANUAL1_CYCLE** via JSR.

MANUAL2_CYCLE_J 62/long
same as **MANUAL2_CYCLE_J**.

MANUAL3_CYCLE_J 66/long
same as **MANUAL3_CYCLE_J**.

MANUAL4_CYCLE_J 70/long
same as *MANUAL4_CYCLE_J*.

MLOOP_PTR 74/long
address pointer; position of the MIDI IN write pointer. Points to the next write address. Read with -1(Ax).

MLOOP_ADR 78/long
address pointer; start of the MIDI IN write buffer.

MFILTER_SWITCH 82/word
flag; if this is <> 0 then the MIDI IN filter is active.

MFILTER_NOTE 84/word
flag; if this is <> 0 the MIDI IN filter lets pass NOTE numbers.

MFILTER_VELO 86/word
flag; if this is <> 0 the MIDI IN filter lets pass NOTE velocities. (if both flags are on note numbers + velocities are written to the buffer).

MSG_PTR 88/long

pointer; points to the personal message buffer in the PATTERNER. By writing a character into the keyboard buffer we can call the *MULTI_LOOP* with a message. The message is in this format: ' $11,<command>' . For commands see list in the **CYCLER** section.

CALL_PTR 92/long

pointer to the keyboard buffer info block. Using data in this block you can send messages into the keyboard buffer and simulate an user-input. See in the *REMOTE* **CYCLE** how it works in detail

INTERNAL_PTR 96/long

pointer to the Internal Sound Module. Use this address within **CYCLER** modules to send data to the Internal Sound Module. This and the following *IX_FLAG* are used in the *MIDI MESSAGE #* and *SEND MIDI MACRO CYCLE* routines.

IX_FLAG 100/byte

this flag holds one byte which describes if the PATTERNER is in MIDI- or Internal Output mode. 0=Internal/<>0=MIDI output. Read-only. In the *MIDI MESSAGE#* and *SEND MIDI MACRO CYCLE* routine this flag is used to decide wether to send the MIDI data to the MIDI Out port to the Internal Sound Module.

7. ...SOMETIMES IT DOES NOT WORK

Whatever happens - don' t blame your ST!

- the program crashes or does crazy things while loading, the MAINPAGE is not even showing up.....???

a) There may not be enough free RAM left in your computer (RAM disk to large or too many accessories?).

b) You may have encountered one of the yet undiscovered program bugs and are urged to re-port to us how you did this. Call (205)-349-3479 24 hours a day, operator could be asleep....

- a **BEAT BANK** has been loaded successfully but there' s no sound when the **BEAT** is played back....???

a) Check if the MIDI keyboard is hooked up correctly

b) Check if the MIDI keyboard is ON

c) Check the volume of your keyboard

d) Check if the **BEAT** has been assigned to an **SCALE** containing usable keynumbers

e) Check the status of the Output Flag (Internal or external?)

f) Check if the MIDI receive channels on the keyboard match with the transmitting channels of the **SCALE**

g) Check that the **BEAT** has more than 0 **TICs**.

h) Check that the **SPEED** the current **BEAT** is using is higher than 0.

- all five components of a **SYSTEM** have been loaded properly, after starting playback of a **BEAT**, a **PATTERN** or a **SONG** there' s no response...???

a) Check the different **BANKS** and their relationships

b) Check the **SYSTEM SPEED**

c) Check the **BEAT INTERNAL SPEED**

- A **CYCLE** program seems to be doing strange things.

a) Don' t be angry! The **CYCLE** system is sensitive and be aware that it is like a programming language. You can do nasty things with it.

b) Get your **CYCLE** back into the **CYCLE** editor. Check in-and outputs - are there values being picked up from the stack before they are put there? (I mean, there are routines checking a lot you do, but also there are loopholes).

c) Plus: The **CYCLE** program is supposed to do strange things!

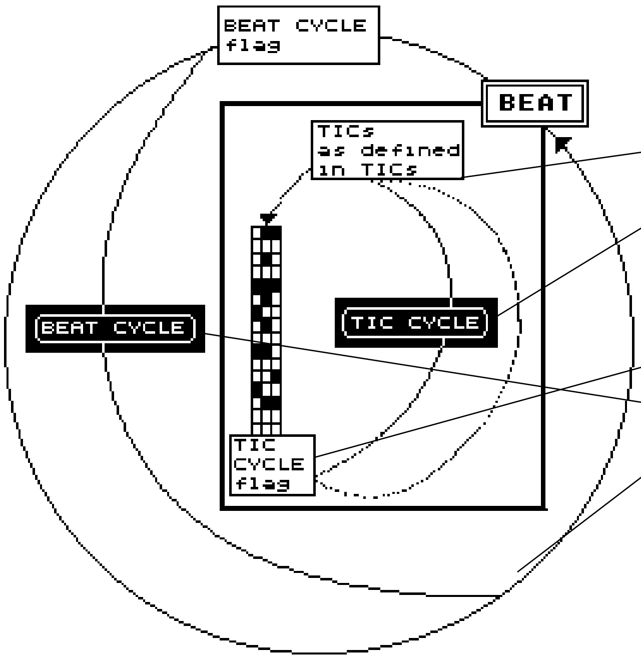
THE CYCLE SYSTEM

C.0. Overview

You probably thought that entering music into PATTERNERs data structures is a lot of work. Agreed! But here' s the reward. The **CYCLE** system is the most innovative feature of the PATTERNER. Like many advanced features it is useful, powerful and a little hard to learn in the beginning.

As you should know by now the PATTERNER uses a different approach in putting together MIDI Messages from the available data **BANKs**. If you don' t, we recommend you read the respective sections in the manual. The **CYCLE** system is built around this different approach. With this you can add your own routines to the playback section of PATTERNER.

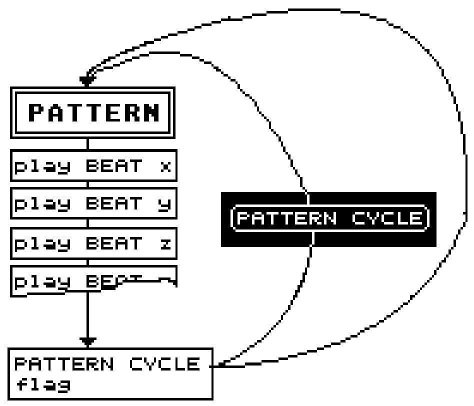
Here a little graphic overview of how **CYCLEs** are linked with the playback routine in the PATTERNER.



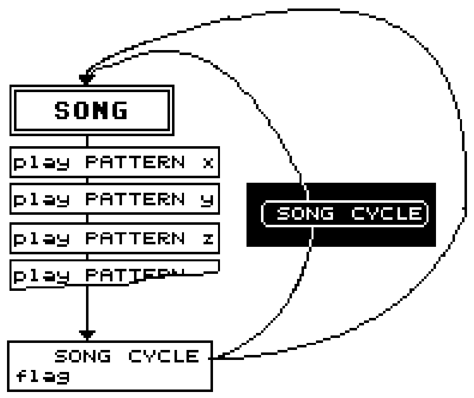
The **TIC** loop represents the innermost loop. One **BEAT** can contain 0-32 **TICs** which are all scanned for Note On/Off messages in ascending order. The first spot for a custom routine is after every played **TIC**. This is the black box called **TIC CYCLE**. The **TIC CYCLE flag** determines if the routine is used or bypassed.

BEATs are the central part of the PATTERNER. The next spot for your own routines is after every played **BEAT**. This is called the **B**EAT CYCLE and the **BEAT CYCLE** flag decides if the routine is used or not.

The upper two **CYCLES** are used whenever playback is running and the respective flag is set. The following two, however, are only used if their flag is set and a **PATTERN** or a complete **SONG** is played back.



A **PATTERN** plays back **BEATs** from the **BEAT BANK** in a pre-programmed order. If the **PATTERN CYCLE** flag is set it also uses the **PATTERN CYCLE** after each finished **PATTERN.**



A **SONG** takes **PATTERNs** from the **PATTERN BANK** and plays those.If the **SONG CYCLE** flag is set it uses the S**ONG CYCLE** after every finished **SONG**.

As you can see in the pictures there are four time spots at which a **CYCLE** can be used, if the respective flag (**FLAGS** menu) is set:

After every played **TIC** -> **TIC CYCLE**
After every played **BEAT** -> **BEAT CYCLE**
After every played **PATTERN** -> **PATTERN CYCLE**
After every played **SONG** -> **SONG CYCLE**

In addition there are four **MANUAL CYCLEs** which are executed whenever their respective key on the ST keyboard is pressed (keys <*1*> to <*4*> on the main keyboard).

Use the **CYCLER Box** to create ready-to-run .*CYC* files on disk. You will find more information on how to do this in the following parts of this book.

In order to run a **CYCLE** file with a special set of **BANKS** you must load all the **BANKS** first (or a complete **SYSTEM** at that). Then load the .*CYC* file on top of that (using *LOAD CYCLES* from *FILE* menu [4.2.]).

Your next step after successful loading will be to enable those parts of the **CYCLE** you want to use. In order to do this, set the respective flags in the *FLAGS* menu [4.4.]. All sections of a **CYCLE** can be switched on or off independently. Launch the **BEAT, PATTERN** or **SONG** and your **CYCLEs** will start working.

C.0.1. TIC CYCLE

If the **TIC CYCLE flag** is enabled the PATTERNER playback routine will run through your **TIC CYCLE** everytime one of the **TICs** in the running **BEAT** has been finished[4.4.4.]. Every active **TIC** (one which is played back) in a **BEAT** runs through the **TIC CYCLE** in this case, no matter if there are routines in it or not. Since **TICs** are the fastest running units in the PATTERNER you should not use **CYCLEs** which are too long or take too much time. This could disrupt the timing structure. You will notice that **TIC CYCLEs** mutate your affected **BANKS** at a pretty high rate.

C.0.2. BEAT CYCLE

If the **BEAT CYCLE flag** [4.4.5.] is set the **BEAT CYCLE** is used after every played **BEAT**. That' s the one I use most.

C.0.3. PATTERN CYCLE

If the **PATTERN CYCLE** flag [4.4.6.] is enabled the **PATTERN CYCLE** is used after every finished **PATTERN** Even empty **PATTERNs** (those containing no BEATs) run through the **PATTERN CYCLE**

C.0.4. SONG CYCLE

If the **SONG CYCLE** flag [4.4.7.] is enabled the **SONG CYCLE** is used after every finished **SONG**. Empty **SONGSs** also run through the **SONG CYCLE**.

C.0.5. MANUAL CYCLES

This single flag controls the status of the four available **MANUAL CYCLEs**. If this flag is enabled the keys *<1>* - *<4>* on the ST keyboard launch the respective **MANUAL CYCLE**. **MANUAL CYCLEs** are also part of the playback routine and thus cannot be called when no playback is in progress.

C.0.6. What does it do in General?

CYCLES can be used to customize the playback routine of PATTERNER.

Basically you use **CYCLER** routines to get the working status of particular values from the PATTERNER (example: **TICs** in current **BEAT, SOUND CHANNEL SELECTION PATTERN** of **SOUND CHANNELs, SYSTEM SPEED**...), then process these values (multiply, add, logical operations, etc...), and then write them back to PATTERNER into the locations they came from or to other locations.

It also lets you remote control some PATTERNER functions. Here an example: You could program a **CYCLE** to *UPDATE* the **MAIN PAGE** everytime a **BEAT** has been finished (same function like clicking over the *UPDATE* button on the **MAIN PAGE**).

PATTERNER also picks up MIDI Messages from the MIDI In port and stores them in an internal buffer. The only way to access this buffer is by using **CYCLE** routines. In this way you are able to program the playback routine to react to what you play on a connected MIDI controller, while playback is running.

For much more detailed information and programming examples, please, consult the following chapters of this section.

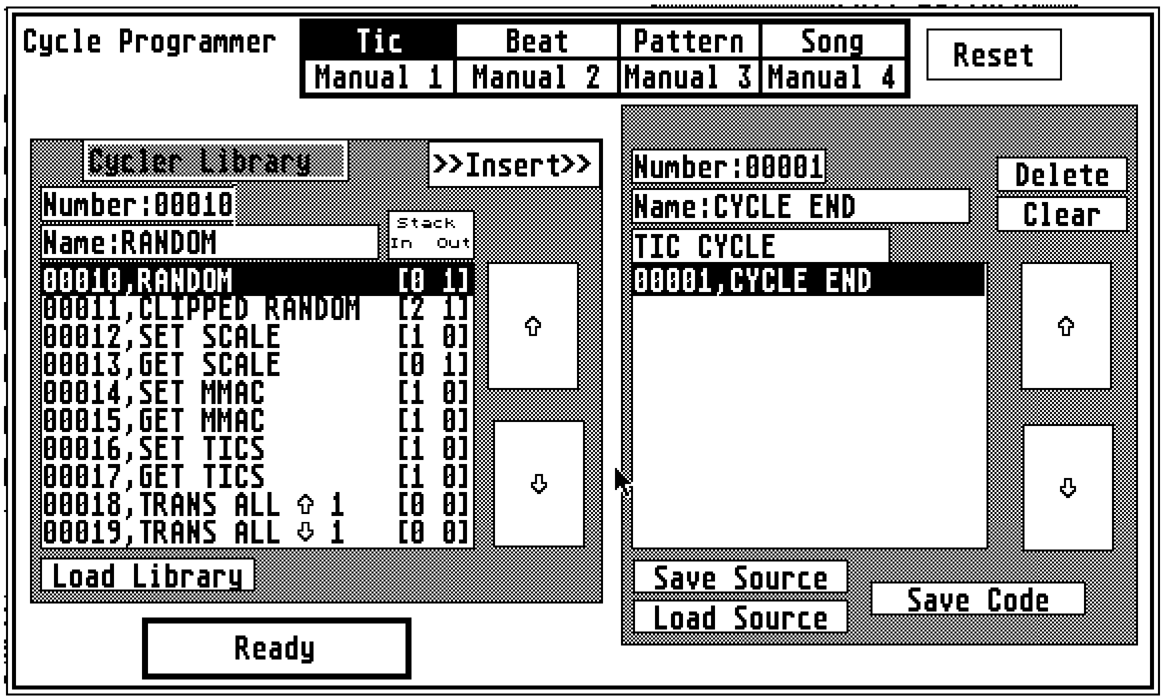
C.1. LIBRARY LEVEL PROGRAMMING

This first part on progamming in the **CYCLE** environment shows you how to use the **CYCLER Box** to put together complete working .*CYC* files from an existing **LIBRARY** of source routines.

C.1.1. Using the CYCLER Box to assemble/edit .CYC files

In order to assemble or edit **CYCLEs** you have to open the **CYCLER Box**. This can be done by either selecting *EDIT CYCLES* in the *FILE* menu of PATTERNER [4.2.7.] or by pressing <*CONTROL C*> on the ST keyboard from the **MAIN PAGE**.

The **CYCLER Box** looks somewhat like this (with **CYCLER LIBRARY** loaded):



```
Number:00010
Name:RANDOM
00010,RANDOM          [0 1]
00011,CLIPPED RANDOM  [2 1]
00012,SET SCALE       [1 0]
00013,GET SCALE       [0 1]
00014,SET MMAC        [1 0]
00015,GET MMAC        [1 0]
00016,SET TICS        [1 0]
00017,GET TICS        [1 0]
00018,TRANS ALL ⇧ 1   [0 0]
00019,TRANS ALL ⇩ 1   [0 0]
```

```
Number:00001
Name:CYCLE END
TIC CYCLE
00001,CYCLE END
```

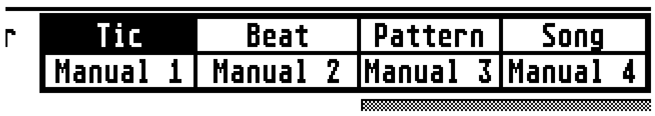
C.1.1.1. **LIBRARY** Programming example 1

In order to explain how to program on the **LIBRARY Level** lets create a simple **CYCLE**. It will be a **BEAT CYCLE** and thus be used everytime after a **BEAT** has been played back. Our B**EAT CYCLE** will transpose up all MIDI Keynumbers of the current **SCALE** whenever it is called.

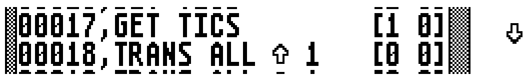
Here' s the step-by-step approach:

0) The first thing you have to do after you opened up the **CYCLER Box** is to load a **LIBRARY** file. Do this by left-clicking over the *LOAD LIBRARY* button below the left window. Select the '*CYCLER.LIB*' file in the following GEM Item Selector. After it has been loaded you see a **CYCLER Box** as shown on the previous page.

1) Since we want to assemble a **BEAT CYCLE** click over the *BEAT* button in the **CYCLE Selection Bar** at the top of the box. This brings up the contents of the current **BEAT CYCLE** (which of course is empty at this time, except for the '*CYCLE END*') in the right window.



2) Next, move the mouse pointer over the '*TRANS ALL arrow up*' entry in the left **LIBRARY** window.



```
00017,GET TICS          [1 0]
00018,TRANS ALL ⇧ 1     [0 0]
```

3) Single left-click over it. This selects this entry by bringing its name into the Number/Name fields above the **LIBRARY** window and draws the entry in the window in reverse video.

4) There are two ways to take over the selected routine into the **BEAT CYCLE** window on the right side:

a) Left-click over the *>>INSERT>>* button above the **CYCLER LIBRARY** window or

b) Double-left click over the entry in the **CYCLER LIBRARY** window.

Now look at the new status of the **BEAT CYCLE**:

BEAT CYCLE (textfield above right window)

TRANS ALL arrow up (New routine inserted and still selected)

CYCLE END

5) Click the *SAVE SOURCE* button in the right window and save your source using a *.SRC* suffix and the GEM Item Selector. Let' s name it *TEST.SRC* for now.

6) Click the *SAVE CODE* button in the right window and create and save the code of the new **CYCLE** using the GEM Item Selector for this and an *CYC* ending. Let' s name this '*TEST.CYC* .

<u>This is already it!</u>

The .*CYC* file is on disk and ready to be used. Click over *READY* or push the <*RETURN*> key on the keyboard to exit the **CYCLER Box**. Then take these steps to hear it :

a) Load one of the demo **SYSTEMs** (or an own **SYSTEM** which must at least contain one **BEAT** in the **BEAT BANK** and one **SCALE** in the **SCALE BANK**).

b) Now, select *LOAD CYCLES* from the *FILE* menu and load in the newly created '*TEST.CYC* file using the GEM Item Selector.

c) Set the **BEAT CYCLE Flag** in the *FLAGS* menu.

d) Start **BEAT** playback by using the **BEAT** playback symbol [3.5.1.] or push the <*B*> key on the keyboard.

If your keyboard is hooked up correctly you will hear your **BEAT** being played back while the keynumbers of the used **SCALE** are transposed up a half step every time the **BEAT** is through.

C.1.1.2. **LIBRARY** Programming example 2

In our second example on **LIBRARY LEVEL** programming we will use the **CYCLE** created in the first example and expand on it. Open the **CYCLER Box** again.

The goal of this **CYCLE** will be to get a random number and then transpose all selected **SOUND CHANNELs** in the current **SCALE** down as many halfsteps as the random number requires, after a **BEAT** has been finished.

0) As preparation for this example you should have loaded the same **LIBRARY** file you used on the first example ('*CYCLER.LIB*') and also have the first example still in the right window (if not, load the file '*TEST.SRC*' which has been created before in Example 1).

1) Click over the **BEAT** button in the **CYCLE Selection Bar** of the **CYCLER Box**. Your old **BEAT CYCLE** will show up in the right window:

```
BEAT CYCLE
TRANS ALL  arrow up
CYCLE END
```

2) Since the '*TRANS ALL arrow up*' entry in the right window is the first entry it will already be selected. Delete this entry by clicking over the *DELETE* button. The **BEAT CYCLE** is empty again. The insertion pointer now is on the '*CYCLE END*' entry - this means that an >>*INSERT*>> you make will be inserted in front of the '*CYCLE END*'.

3) Move the mouse pointer into the left **LIBRARY** window over the <*down arrow*> button. We are looking for a routine called '*TRANS SEL arrow down*', which is not available from the **LIBRARY** window as it is now. Single-click over the <*arrow down*> button to scroll through the **LIBRARY** until you see the requested entry.

4) Found it! Double-left click over it and see how it is copied over to the **BEAT CYCLE**. That's good.

5) Here comes a programmed mistake (to show how it works):

a) Save the **CYCLER** source with *SAVE SOURCE* as '*TEST.SRC*'

b) Try to save the **CYCLER** code with *SAVE CODE*....

c) "*Not enough values on stack...*"

This is the error message you will see now!

Look at the '*TRANS SEL* arrow down' entry in the **LIBRARY** window more closely and you will notice that there are two numbers written behind this routine name. The first number ('1') tells us that this routine requires one value on the stack before it can be called (one value coming IN). The second value ('0') tells us that it doesn't change the stack after it is done (no values going OUT).

Here's how our BEAT CYCLE looks right now:

```
BEAT CYCLE

TRANS SEL arrow down
CYCLE END
```

Since the '*TRANS SEL arrow down*' routine requires one value from the stack but no values have been put there we will naturally get an error message!

6) Let's create the missing stack value. Double-left click over the <*arrow up*> button in the **LIBRARY** window to get to the very beginning of the **LIBRARY** where you find the '*RANDOM*' routine.

7) Back to the **BEAT CYCLE** window. Left-click over the '*TRANS SEL arrow down*' entry to move the insert pointer right in front of it (it must be selected now!).

8) Double-left click over the '*RANDOM*' routine entry in the **LIBRARY** window to copy it over into the **BEAT CYCLE**. This is what the **BEAT CYCLE** looks like now:

```
BEAT CYCLE

RANDOM
TRANS SEL arrow down
CYCLE END
```

9) Now again, save both **SOURCE** and **CYCLE** files as '*TEST.SRC*', '*TEST.CYC*' respectively. This time there shouldn't be any error messages.

Exit the **CYCLER Box** and proceed like you did in <u>Example 1</u> to check your new **CYCLE**. Only selected **SOUND CHANNELs** are transposed by this **CYCLE**, so be sure to select some [3.2.] on the **MAIN PAGE**.

10) Also modify your example '*TEST.SRC*' to make it look like:

```
BEAT CYCLE

GET CHANNELS             use the DELETE button to get rid of 'RANDOM'
TRANS SEL  arrow down
CYCLE END
```

You will probably not hear too many differences. But notice how we use the current **CHANNEL SELECTION PATTERN** to derive the transpose value (which is kind of backwards, since the **CHANNEL SELECTION PATTERN** determines which **SOUND CHANNELs** are transposed and which not).

11) To screw up things even more try this one:

```
BEAT CYCLE

GET CHANNELS
NOT CHANNELS
TRANS SEL  arrow down
CYCLE END
```

Experiment with the available material. After you tried your **CYCLEs**, always reload the complete **SYSTEM** before running again - data are changed by **CYCLES**!

12) Here some more examples. Don' t limit yourself to the **BEAT CYCLE**. You can also use the **TIC CYCLE** or any other one (if your **SYSTEM** contains **PATTERNs** and **SONGs**).

a)

```
any  CYCLE
GET CHANNELS          (write selection pattern to stack)
SET SYSSPEED          (pick value from stack and use it to set speed)
CYCLE END
```

b)

```
TIC CYCLE
GET SYSSPEED          (write system speed value to stack)
SET CHANNELS          (pick value from stack and use it to set selection)
ROL CHANNELS          (rotate left current selection pattern value)
CYCLE END
```

```
BEAT CYCLE
GET CHANNELS          (write selection pattern to stack)
SET TICS              (pick up value from stack and use to set tics)
CYCLE END
```

C.1.1.3. **LIBRARY** Programming example 3

If you use *RANDOM* number or other inputs you may not always get values within a usable range.

Look at this **CYCLE**:

```
any  CYCLE
GET CHANNELS      (write selection pattern to stack)
SET TICS          (pick up stack value and use to set tics)
CYCLE END
```

First, we *GET* the **CHANNEL** Selection Pattern. This may be a very high or very low value. You don' t know....

Since the '*SET TICS*' routine clips all incoming values to ' 32' or lower there won' t be problems with out of range **TIC** values. But it may also clip off the interesting part of a value.

Here' s an alternative:

```
any  CYCLE
GET CHANNELS      (write selection pattern to stack)
DIVIDE 00000      (appears as 'DIVIDE #' in the LIBRARY)
SET TICS          (pick ap value from stack to use as TICS)
CYCLE END
```

We still put our **CHANNEL SELECTION PATTERN** to the stack. But here we process it. It can be divided by any number we choose before we use it as new **TICS** value.

1) Double-click over the '*DIVIDE 00000*' entry in the **CYCLE** window. This brings up the **VALUE Enter Box**.

2) Enter a sense-making value (maybe 6 or something). You can enter decimals here (like 450) or hexadecimals (like $ff5) or even binaries (like %110011). Keep in mind though, that this is an integer division with no remainder.

3) Click *OK* to take the value over (omit zero (' 0') to avoid a dangerous ' division by zero' error which the 68000 processor doesn' t like).

4) Now the **CYCLE** looks like this:

```
any  CYCLE
GET CHANNELS        (write selection pattern to stack)
DIVIDE 00006        (divide uppermost stack entry by 6)
SET TICS            (pick up stack value to set tics)
CYCLE END
```

If you use the **VALUE Enter Box** to input numbers you may enter them in one of the three number systems. They are displayed, however, in decimal at all times!

C.1.1.4. **LIBRARY** Programming example 4

This following example will use one of the more advanced routines of the **CYCLER LIBRARY**. Now, that you are familiar with the process of editing **CYCLEs** here's how our next one should look like:

```
TIC CYCLE

PUSH 00064          (write the value 64 to the stack)
PUSH 00060          (write the value 60 to the stack)
PUSH 00144          (write the value 144 to the stack)
MIDI MSG 00003      (pick up the last 3 stack entries and send to MIDI out port - last
                      in/first out: meaning 144, 60, 64)
CYCLE END
```

You will understand the first three routines which simply write three values on the stack.

The '*MIDI MSG #*' routine is very interesting and can be used in many ways. In our example we use it to send a NOTE On message to the keyboard. Here it is important to know that the stack is a Last In/First Out. This means that the values are *PUSH*ed in this order: 64, 60, 144; and then picked up like this: 144, 60, 64!

'*MIDI MSG 00003*' will try to pick up three values from the stack and send them to the MIDI Out port in the pick-up order. In this case the values are: 144 (decimal for $90), NOTE On Message on channel 1; 60, Keynumber of note we want to hear; and 64, Velocity of the note. In case the PATTERNER is switched to Internal Sound (*DEFAULTs* menu) '*MIDI MSG #*' sends the data to the Internal Sound Module.

Another example:

```
TIC CYCLE

PUSH 00064          (velocity note 2)
PUSH 00060          (keynumber note 2)
PUSH 00144          (Note On status byte note 2)
PUSH 00064          (velocity note 1)
PUSH 00064          (keynumber note 1)
PUSH 00144          (Note On status byte note 1)
MIDI MSG 00006      (send the last 6 stack entries to the MIDI out)
CYCLE END
```

Naturally you can send all kinds of MIDI Messages (wether they make sense or not - they may confuse your keyboard, though). If you want to send more bytes than available through the stack you may have to use multiple '*MIDI MSG#*' routines like this:

```
TIC CYCLE

PUSH 00000          (All Notes Off parameter 2)
PUSH 00123          (All Notes Off parameter 1)
PUSH 00176          (All Notes Off status bytes)
MIDI MSG 00003      (pick up three entries from stack and send)
PUSH 00064          (velocity note 2)
PUSH 00060          (keynumber note 2)
PUSH 00144          (Note On status byte note 2)
PUSH 00064          (velocity note 1)
PUSH 00064          (keynumber note 1)
PUSH 00144          (Note On status byte note 1)
MIDI MSG 00006      (pick up last six stack entries and send)
CYCLE END
```

Here the first '*MIDI MSG #*' sends an ALL NOTES OFF Message for channel 0 (this should stop all stuck notes) and the second '*MIDI MSG #*' then sends two NOTE On Messages.

The previous example which sent nine values to the MIDI port could also be done in a totally different way:

```
TIC CYCLE

PUSH  no of MIDI Macro     (this MIDI Macro contains the nine bytes)
SEND MIDI MACRO            (send this complete MIDI Macro to the MIDI Out port)
CYCLE END
```

As you can see this requires a lot less code. The first routine writes the number of the **MIDI Macro** containing the respective MIDI data to the stack. '*SEND MIDI MACRO*' picks the value up and sends the contents of the respective **MIDI Macro** to the MIDI out port, or to the Internal Sound Module (depending on the status of *IX_FLAG* or *MIDI/Internal Output flag* in *FLAGS* menu [2.5.] and [6.11]).

To conclude this section try this:

```
TIC CYCLE

PUSH 00064        (write 64 on stack)
PUSH 00060        (write 60 on stack)
MDI MSG 00003     (try to pick up three values...)
CYCLE END
```

I am sure you have already discovered the obvious mistake: We are writing two values to the stack and then try to pick up three. But if we run the *SAVE CODE* option there is no error message (although the un-detection of this can very easily crash the PATTERNER!).

The '*MIDI MSG #*' routine switches off the stack control feature for every **CYCLE** it is used in. This has to be done in order to make the multiple input feature possible. So, be aware to keep an eye on those stack in/outputs.

Be aware that the **CYCLER**'s internal stack may never exceed ten entries. The stack error trapping will usually detect an overflow but not in **CYCLEs** using the '*MIDI MSG #*' routine.

C.2. ASSEMBLY LEVEL PROGRAMMING

Although the current **CYCLER Routine Library** ('*CYCLER.LIB*') already contains a lot of routines, you may be missing just the one you need for an exciting project. There are two ways out of this dilemma:

1) Drop us a message about what you would like your routine(s) to do. We will then try to do our best to extend the existing **LIBRARY** file by your suggested routines.

2) Or extend the **LIBRARY** yourself. To do this you need any 68000 assembler which can save code as binary file (non executable object file) and of course as much knowledge about assembly language as possible.

The usage of assembly language for this purpose is required because **CYCLER** code is executed within the playback interrupt loop. For this reason the code has to be as fast and compact as possible.

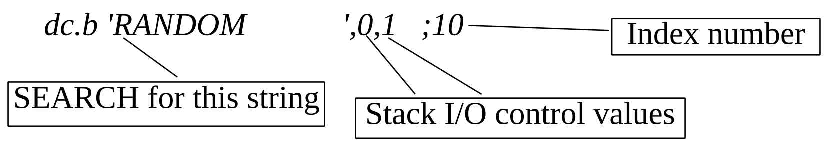
On the program disk you find a file called '*CYCLER.LIB*' containing all the routines described in the accompanying **CYCLER Routine Reference**. This file was created by assembling the '*CYCLER.Q*' file (also on the program disk) and saving the code as binary file.

The source code ('*CYCLER.Q*') has been created with the ASSEMPRO assembler but should work without (m)any adjustments with any assembler capable of saving object files. (i.e. the following: GST ASM, K-SEKA, Hisoft Devpac, Metacomco, Digital Research. Also Public Domain assemblers are available).

You also need information about some internal data formats and arrays of the PATTERNER, which you can address using **CYCLER** routines. This information can be found in the section about file formats [6.11] in compiled format or also spread out in the following text.

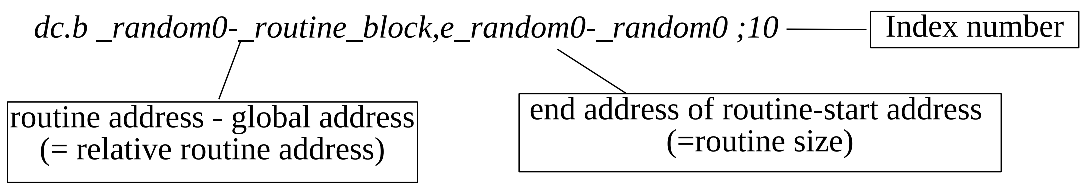
C.2.1. Alter existing routine

To alter the contents of one single routine load the '*CYCLER.Q*' file into the editor of your assembler. Scroll through the source text until you have the very routine on screen. If you have problems finding the routine's code: Use the SEARCH string feature of your editor to search for the name string of the routine you want ('*RANDOM*' or '*SET SCALE*', etc.). The text line you find will look like this:



This is a spot in the '*_routine_names*'-list, where the names, stack I/O's and ID numbers for all existing routines are defined. The ID number of the routine we're looking for is '10'.

Next, find the '*_routine_block*'-list label. Those entries look like this:


dc.b _random0-_routine_block,e_random0-_random0 ;10
Index number
routine address - global address (= relative routine address)
end address of routine-start address (=routine size)

As you can see, entries in this list also carry corresponding ID numbers. The first longword ('*_random0*' in this case) is the label name where we finally find the code of the routine. You can again use your SEARCH feature for that.

Change, whatever you want to change within the routine's source and assemble the new file.

After successful assembly don't forget to save both source.*Q*) and destination (.*LIB*). It is important that you save the destination as object file with no header.

When doing this kind of work never work with the files contained on the original program disk. In fact we strongly recommend that you only alter copies of your original **LIBRARY** source (.*Q*) -and code (.*LIB*) files and not the original. This helps keeping you out of trouble in case something goes wrong (which most likely is the case with this kind of thing).

C.2.2. Add new routine

If you want to add your own routines to the **CYCLER LIBRARY** you must follow these guidelines. In this example we will append one new routine to the **LIBRARY** file.

Ok. Here are the PATTERNER specific steps:

1) Load your assembler program (or an ASCII text editor, if your system uses separate modules).

2) Load the **LIBRARY** source text, '*CYCLES.Q*'.

3) Find this spot in the source text (use the SEARCH feature of your editor):

```
;-_ROUTINE_NAMES----------------
_routine_names:
dc.b '  CYCLE INIT          '  ,0,0 ;0
dc.b '  SLOT CYCLE          '  ,0,0 ;1
dc.b '  BEAT CYCLE          '  ,0,0 ;2
dc.b '  PATTERN CYCLE       '  ,0,0 ;3
dc.b '  SONG CYCLE          '  ,0,0 ;4
dc.b '  MANUAL 1 CYCLE  '  ,0,0 ;5
dc.b '  MANUAL 2 CYCLE  '  ,0,0 ;6
dc.b '  MANUAL 3 CYCLE  '  ,0,0 ;7
dc.b '  MANUAL 4 CYCLE  '  ,0,0 ;8
dc.b '  CYCLE END           '  ,0,0 ;9
dc.b '  RANDOM              '  ,0,1 ;10
dc.b '  CLIPPED RANDOM  '  ,2,1 ;11
dc.b '  SET SCALE           '  ,1,0 ;12
dc.b '  SET MMAC            '  ,1,0 ;13
dc.b '  .........
```

This is the start of the '*_routine_names*' -list. It contains the name strings of all routines as they appear in the **CYCLER Box LIBRARY** window. The two numbers behind every name represent the stack input/output check values. They also show up in your **LIBRARY** window. Ok, so entries into the '*_routine_names*' -list use this format:

dc.b '<name>',i,o ;ID

<name> must be a sixteen character textstring which may use all available characters. Fill up with blanks (*<SPACEBAR>*) when necessary.

'*i*' is stack input control value which must correspond with the values your routine tries to pick up from the stack.

'*o*' is the stack output control value. This must correspond with the number of values your routine writes to the stack.

'*ID*' simply represents the routine' s relative position. It is just a comment (because of ' ;')-but all entries should be ID' ed to avoid confusion!

You should use very informative names for your routines!

Since we want to append a new routine to the existing **LIBRARY** we have to find the last entry in the '*_routine_names*'-list:

```
dc.b '  MIDI IN exp         '  ,0,1 ;61
dc.b '  REMOTE exp          '  ,1,0 ;62
dc.b '  SEND MIDI Macro '  ,1,0 ;63
;------------------------------------------------
```

In the **CYCLER LIBRARY** source file which comes on your disk '*SEND MIDI Macro* ' is the last routine.

Our new routine will just do one simple thing: Look up the number of the currently played TIC and write it to the stack. Therefore our '*_routine_names*' -list entry looks like this:

dc.b 'GET CURR TIC   ',0,1 ;64

As you can see, the one value it will write to the stack is reflected in the stack I/O control bytes. Insert this line just after entry 63:

```
dc.b '  MIDI IN exp          '  ,0,1 ;61
dc.b '  REMOTE exp           '  ,1,0 ;62
dc.b '  SEND MIDI Macro '  ,1,0 ;63
dc.b 'GET CURR TIC      ',0,1 ;64
;------------------------------------------------
```

4) Next, continue to move through the **LIBRARY** source text until you find the '*_routine_block*' -label. This is where the routine pointers and sizes are created when you assemble the file. Find this spot in the '*routine-block*'-list:

```
 dc.l _midi_in-_routine_block,e_midi_in-_midi_in              ;61
 dc.l _remote-_routine_block,e_remote-_remote                 ;62
 dc.l _send_mmac-_routine_block,e_send_mmac-_send_mmac ;63
e_routine_block:
```

Insert the following text line after the '*_send_mmac...*' entry and in front of the '*e_routine_block*' -label:

dc.l _get_curr_tic-_routine_block,e_get_curr_tic-_get_curr_tic   ;64

Now you should get this picture:

```
 dc.l _midi_in-_routine_block,e_midi_in-_midi_in                 ;61
 dc.l _remote-_routine_block,e_remote-_remote                    ;62
 dc.l _send_mmac-_routine_block,e_send_mmac-_send_mmac   ;63
 dc.l _get_curr_tic-_routine_block,e_get_curr_tic-_get_curr_tic ;64
e_routine_block:
```

It is very important that the relative position of the name entry in the '*_routine-names*' -list corresponds with the relative position of the pointer entry in the '*_routine-block*' -list. So, please, do use the ID numbers!

5) Now we are ready for the last major step: adding the code of the routine. Theoretically, this code can be located anywhere beyond the '*_routine-block*' . It is very practical, however, to keep the routine codes in the same order like their entries in the name and pointer lists.

The last routine in the **LIBRARY** file is '*_send_mmac*' and you' ll find it at the very end. You can start to enter your new routine source text right after the '*e-send-mmac*' label, which terminates the previous routine. It is a good idea to put some comments in front of the new routine (stuff like what the routine does, how many parameters it wants, it' s ID, etc..).

Okay, here is our new routine:

```
;GET_CURR_TIC ;64
; reads the number of the currently played TIC from
;the playback routine and writes it on the stack
_get_curr_tic:
  move.l WORK_LIST(a6),a0
  move.w TIC_COUNTER(a0),(a5)+
e_get_curr_tic:
```

Look at the two used labels '*_get_curr_tic*' and '*e_get_curr_tic*' . Both are needed in the '*_routine_block*' -list to create the pointers and sizes.

In the first row of code (*move.l WORK_LIST(a6),a0*) we get the global pointer to the playback array in A0. The second row reads the needed value (TIC_COUNTER) from the array and writes it onto the stack. [see 6.11. for details on the Internal playback array].

6) Assemble the code, and if successful (no errors) save it as binary file. Use a *.LIB* ending for this file (If you don' t, the **LIBRARY** will not be displayed in the GEM Item Selector when you want to *LOAD* a **LIBRARY** but can still be loaded).

7) Your library file can now be loaded with *LOAD LIBRARY* within the **CYCLER Box**. All routine names beyond '*TRIGGER END*' should be displayed in the **LIBRARY** window.

<u>If not all routine names are displayed or if there are other irregularities (strange text effects...) check your new **LIBRARY** routine entry following the steps from number 1!</u>

'Add routine'-checklist:

A) Append the name string of your new routine to the '*_routine_names*'-list. (see 1 - 3)

B) Append the pointer entry of your new routine to the '*_routine_block*'-list. (see 4)

C) Append your routine's code at the file end. (see 5)

D) Save, assemble and find out!

Important:

Relative positions of entries in the '*_routine_names*'-list and the '*_routine_block*'-list must be identical. Use ID numbers in comments to make sure!

No **CYCLER** routine must access other PATTERNER locations than the ones defined by the *Internal playback access array* [6.11.].

You can reserve memory with the 'dc.x' assembler directive within a routine or you might use the processor stack (movem.l ...). By the time you exit the routine, however, the stack must be in the same state as it was before.

The following **CYCLES** are carried out in an interrupt:

TIC CYCLE, BEAT CYCLE, PATTERN CYCLE and **SONG CYCLE**

DON'T use 'TRAP #<n>' calls in there! But you can access restricted memory areas since we are in Supervisor mode.

The **MANUAL CYCLES** may use TRAP calls and therefore can be used to write strings to the screen or whatever.

Recommended books to read about the ATARI's Operation System:
ST Internals from Abacus
The Concise ATARI ST 68000 Programmer's Guide from Glentop

C.2.3. Exceptions and additional features

The previous steps describe how to alter existing routines in the '*CYCLES.Q*' **LIBRARY** source file or how to add new routines to it. In order to put this programmability to full work you must know as much as possible about 68000 assembly language.

C.2.3.1. Routines using the Value Enter Box

If you have already gone this far in this manual I assume you know that the **CYCLER LIBRARY** contains several routines which accept number values from the keyboard ('*PUSH #*', '*AND #*'...) before saving their *CODE* to disk..

Read in the **CYCLER References** chapter [C.3.] on how this **Value Enter Box** is used. In the following you can find out how routines have to look like if they want to use this feature. See previous chapter on how to append your own routines to the **LIBRARY** in general.

Here are the alterations you have to make in order to use the **Value Enter Box**:

A) '*_routine_names*' -list entry

regular: *dc.b '<routine name> ',i,o ;ID*

new: *dc.b '<routine name>*<u>*#*</u> *',*<u>*$80+*</u>*i,o ;ID*

1) the last character of the name string must be ' #' !
While the name is displayed with the ' #' character in the **LIBRARY** window, the ' #' is replaced by a five digit value (' 00000') in the **CYCLE** window. Watch it! The ' #' is a placeholder for five digits and you must leave enough space behind it in the string and still not exceed the 16 character limit.

2) by adding '*$80*' to the input control value you set the **Value enter flag**
In other words: if bit 7 in the stack input control value is clear you' re dealing with a regular routine. Is bit 7 set the **Value Enter Box** can be used.

<u>Both, '*#*' character AND '*$80*' are needed to use the **Value Enter Box**!</u>

B) There are NO changes in the '*_routine_block*' -list entry

C) In order to process the incoming value (the one from the **Value Enter Box**) the first line of your routine must look like this:

move.w #0,Dx

At *SAVE CODE* time the entered value will replace '*#0*' (#<new value>). '*Dx*' stands for any of the available data registers of the 68000.

Every routine can only accept one value from the **Value Enter Box** and the value has to be processed in the above mentioned manner. The **Value Enter Box** accepts numbers in either decimal, hexadecimal ($<number>) or binary (%<number>) formats.

C.2.3.2. Switch off the Stack Control feature

If you try to save *CYCLER CODE* files (.*CYC*) from the **CYCLER Box** you actually launch the assembler run, which links all needed routines into a ready-to-run **CYCLER** file.

In order to prevent you from making mistakes by pushing too many (or not enough) values onto the **CYCLER** stack, every **CYCLE** is checked for stack mismatches. If there are none, the file is in fact created and saved. If there are errors you' ll get one of several error messages (as described in the *APPENDIX*).

It is possible to switch off this stack-error trapping feature in which case you can do to the stack whatever you want without getting any error messages. This is used in the '*MIDI MSG #*' routine. The routine picks up a value with the **Value Enter Box** and uses this value to determine, how many values have to be picked up from the stack to be sent out to the MIDI port. So in this case the stack input control value is a variable!

To switch off the stack-error trapping, change the '*_routine_names*' -list entry of the respective routine like this:

regular: *dc.b '<routine name> ',i,o ;ID*

new: *dc.b '<routine name> ',i,$80+o ;ID*

The **CYCLER** *SAVE CODE* option will check bit 7 in the stack output control value to decide if the stack must be checked for the respective **CYCLE** or not. Set bit 7 ('*$80+o*') to set the flag and bypass the error trapping.

All other **CYCLES** are still checked as usual. As you can see in the '*MIDI MSG #* ' routine you can do both things in the same routine: Use the **Value Enter Box** AND switch off the stack control.

All routines which shut off the error trapping appear like this in the **LIBRARY** window:

<ROUTINE NAME> x,x

C.2.4. CYCLER Library Assembler Source

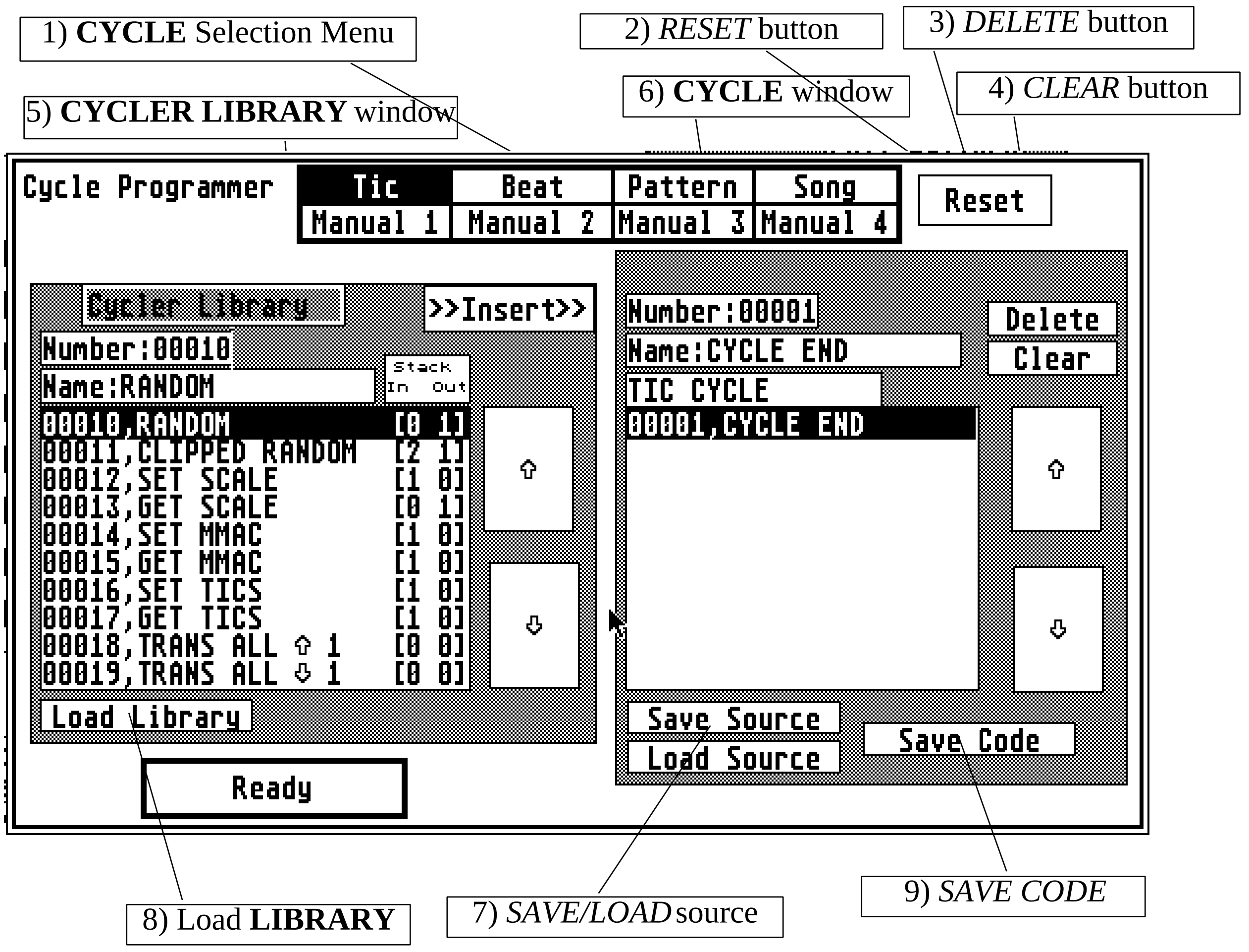
You find the assembly source code of the current **CYCLE LIBRARY** on your program disk as '*CYCLER.Q*' . Accompanying this manual you also get a printout of this file and a reference of all routines and what they do.

This information comes seperate from the manual because it will have to be updated in short intervals (as the **LIBRARY** evolves).

C.3. CYCLER References

C.3.1. CYCLER Box reference

Here a description of all **CYCLER Box** components.



```
Number:00010
Name:RANDOM
00010,RANDOM         [0 1]
00011,CLIPPED RANDOM [2 1]
00012,SET SCALE      [1 0]
00013,GET SCALE      [0 1]
00014,SET MMAC       [1 0]
00015,GET MMAC       [1 0]
00016,SET TICS       [1 0]
00017,GET TICS       [1 0]
00018,TRANS ALL ⇧ 1  [0 0]
00019,TRANS ALL ⇩ 1  [0 0]
```

```
Number:00001
Name:CYCLE END
TIC CYCLE
00001,CYCLE END
```

1) **CYCLE** Selection Menu

Here you can select one of the eight different available **CYCLEs**. The contents of whichever **CYCLE** you select are displayed in the right **CYCLE** window.

A complete **CYCLE** file (.*CYC*) consists of all eight **CYCLEs**, each containing at least the '*TRIGGER*' (which is always displayed in its own extra small text window above the **CYCLE** window) and the '*CYCLE END*' (displayed IN the **CYCLE** window at the last position), both of which are not delete-able.

2) *RESET* button

This brings up a warning. If this operation is executed, all eight CYCLEs are reset to their minimum contents ('*TRIGGER'+'CYCLE END*'). In other words you loose all **CYCLES** in memory.

3) *DELETE* button

This removes the selected entry in the currently active **CYCLE** without warning. The '*CYCLE END*' routine cannot be removed even if it is selected - in this case nothing happens.

4) *CLEAR* button

This resets the currently selected **CYCLE** only after a safety request. Only the '*TRIGGER*' and '*CYCLE END*' routines are left.

5) **CYCLER LIBRARY** window

The source window displays an excerpt from the complete available **CYCLER LIBRARY** you have loaded. Display starts from number ten. Numbers below ten are not accessible (those contain all '*INIT*', '*TRIGGER*' and '*CYCLE END*' routines which are installed automatically when using *SAVE CODE*).

To the right of every individual **LIBRARY** entry you see two bracketed numbers. The first tells you how many inputs the routine expects from the **CYCLE** stack and the second how many outputs it writes to the stack. If no numbers are shown ('*x x*' instead) the respective routine doesn't use the Stack Control feature in whichever **CYCLE** it is used.

NOTE:
Whenever you use the *SAVE CODE* option to create an executable .*CYC* file you will be using routines from an available **CYCLE LIBRARY**. In order to be able to edit the source file (.*SRC*) at a later time always save both .*CYC* and .*SRC* files. Also do never load a .*SRC* file before a **LIBRARY** has been installed!

1) Use the arrow buttons to scroll through the **CYCLER LIBRARY** window. Double-click over an arrow button brings you to the first/last **LIBRARY** entry. Scrolling does not affect the currently selected entry.

2) Select any visible entry by clicking over its number/name in the **LIBRARY** window. This brings the number and name of the new selection into the Number/Name fields just above the **LIBRARY** window. In addition the new selection is displayed in reverse video.

3) You can also use the Number/Name fields just above the **LIBRARY** window to select routines.

To do this, bring the cursor into the required text field using either the <*CURSOR*> keys or single left-click over it with the mouse. Press <*ESC*> to delete the former contents of the field. Enter the number of the routine you need into the Number field and double-click over it to start a number search. Or enter the first characters or the full name of the routine you need into the Name field and double-click over it to start a name search. The found entry is the new selected entry.

You can' t access the *CYCLER INIT* routines. Numbers or names which try to find *INIT* routines will select the first available entry (10, '*RANDOM*'). The name search looks for the name with the highest account of matching characters. If a number is too high, or no name match has been found the last **LIBRARY** entry is selected.

4) Use the >>*INSERT*>> button to insert the currently selected **LIBRARY** entry into the current **CYCLE** destination structure.

Or double-click over the desired **LIBRARY** entry to *INSERT*.

No changes can be made to the **CYCLER LIBRARY** in this level. Thus, there are no edit buttons available for this purpose.

6) **CYCLE** window

Here you can see a part of the contents of the currently selected **CYCLE**.

Each **CYCLE** starts with its '*TRIGGER*' routine - this '*TRIGGER*' is not accessible and displayed in its own little text field just above the **CYCLE** window. The minimum contents of a **CYCLE** are a '*TRIGGER*' routine and a '*CYCLE END*' routine.
If you *SAVE* the *CODE* of such an "empty" **CYCLE** to disk and load it to use with the PATTERNER it won' t do anything, except initialize a few pointers, save the registers to the system stack and restore the registers. After that it will return properly to its caller within the PATTERNER.

1) Like in the **CYCLER LIBRARY** window use the arrow buttons to scroll through the contents of the selected **CYCLE**. Double-click an arrow button to access the start/end of the selected **CYCLE**.

2) Click on any number/name display in the **CYCLE** window to select the entry. The current selection is displayed in reverse and also shown by its number and name in the Number/Name fields just above the **CYCLE** window.

The selection in the **CYCLE** window provides your edit pointer. This means that a *DELETE* operation removes just this selected entry (all following entries move one notch). Also, new entries from the **CYCLE LIBRARY** are *INSERTED* in front of this position. Inserting a new entry moves all following entries one notch to make room. It also moves the pointer (selected entry).

Entries in the **CYCLE** window can be double-clicked if they require the input of a value using the **Value Enter Box**. If a routine's name does not include a '#', double-click has no effect.

Routines which require input appear with a '#'-sign after their name in the **CYCLER LIBRARY** window and are displayed with '*00000*' right after being *INSERTED* into the **CYCLE** window. After double-clicking the entry in the **CYCLE** window you can enter a decimal/hexadecimal/binary value using the **Value Enter Box**.

Decimal numbers are entered just as they are (example: 450, 23...), hexadecimals must start with a '$' header (example: $56, $a0...), and binary numbers must have a '%' header and only contain '0's and '1's (example: %00110011). Click *OK* to take the value over into the routine or *CANCEL* to abort. After clicking *OK* the inserted value is displayed as five digit decimal within the routine name in the **CYCLE** window. Values up to 65536 (16 bit) are available. Values higher than this are truncated to 16 bit.

7) *SAVE/LOAD SOURCE*

A) *SAVE SOURCE*

CYCLER source data is kept in memory as a simple codified string of numbers [see chapter 6. on data formats]. *SAVE SOURCE* lets you save the current data string. **CYCLER** source files are using a *.SRC* suffix.

A *.SRC* file doesn't contain any names or code. All routines are represented by ID numbers according to the **LIBRARY** file they were inserted from. If you create and save a *.SRC* file using one **CYCLER LIBRARY** and load it back under another **LIBRARY** the ID numbers may not correspond with the same routine names. *SOURCE ID* numbers correspond to the relative position of the entry within the **LIBRARY** they come from.

B) *LOAD SOURCE*

This loads raw **CYCLER** data from disk using files with an *.SRC* suffix.

All current contents in the **CYCLEs** are replaced by the new data. *SAVE/LOAD SOURCE* is the only opportunity to save **CYCLER** files AND be able to edit them at a later time. Processed **CYCLER** files cannot be re-loaded or changed (except with an assembler/debugger and a lot of calculating). So be sure to save **CYCLEs** in *.CYC* as well as in *.SRC* form, and know which **LIBRARY** they were created with.

8) Load **LIBRARY**

With this button you can load a new **CYCLER LIBRARY** file. The **LIBRARY** currently installed will be lost as well as all **CYCLES** created with it. There is one **LIBRARY** file supplied on the program disk ('*CYCLER.LIB*'). Data formats and how to create your own **LIBRARIES** are described in the *Assembly Level Programming* section of this manual. Successfully loading always resets all **CYCLER** contents.

9) *SAVE CODE*

This launches the actual **CYCLE** assembly run. **CYCLER LIBRARY** routines are put together to create the complete **CYCLER** program code according to the data in the current **CYCLES**. This option takes all necessary steps, including the creation of an external reference header and checking of **CYCLE** internal stack outputs and inputs.

Most **CYCLER** routines work intensively with the **CYCLE** stack. It is absolutely necessary that all values delivered to this stack are picked up and that all values other routines want to pick up are found. Only if there are no stack errors you will be able to save a complete **CYCLE** file. If there are too many inputs or outputs on the stack you will get one of three error messages:

a) *"Not enough values on stack or wrong input/output order in:<CYCLE NAME>"*

If you get this error message during a *SAVE CODE* operation one or more of the routines in the respective **CYCLE** try to pick up values from the stack which are not there (and thereby decreasing the stack pointer below zero).

Another reason may be that a routine tried to pick up a value from a still empty stack (stack underflow).

b) *"Stack overflow in:<CYCLE NAME>"*

This error message tells you that the routines in the respective **CYCLE** try to write more than the maximum of ten values to the stack.

c) *"Stack input/output unmatch in:<CYCLE NAME>"*

If you get this error the number of inputs and outputs on the stack does not match in the shown **CYCLE**.

Refer to the descriptions of individual **CYCLER** routines for their input/output values [C.3.2. CYCLER LIBRARY] or get this information in the **CYCLER LIBRARY** window left to each **CYCLE** routine name, where stack inputs/outputs are listed.

The **CYCLER** does not check if values are really written to the stack BEFORE they are picked up, except on a stack underflow.

Always keep in mind that you are dealing with a Last In/First Out stack. So be sure to deliver the value which has to be picked up last as the first one.

Note: There is one routine in the **LIBRARY** which bypasses the stack error trapping. This is the '*MIDI MSG #*' -routine. It has been programmed to reset the stack control register. This was necessary because this routine can have different numbers of input values determined by the direct value ('*#*').

C.3.2. CYCLER LIBRARY Reference

Since this reference chapter will be updated in pretty short intervals it comes apart from the manual.

About the Tutorials

The PATTERNER is composed of many different sections. Despite the effort in this manual to explain all functions as comprehensible as possible you may still not know exactly how to start working with it.

The following Tutorials are intended to help you with this. While they are far from explaining everything you can do they will introduce you to the very basics and then let you expand from there.

TUTORIAL 1

This first Tutorial will show you how to create a C-Major **SCALE**. Then we will play the **SCALE** back. In addition the used structures will be named and then saved.

Preparations:

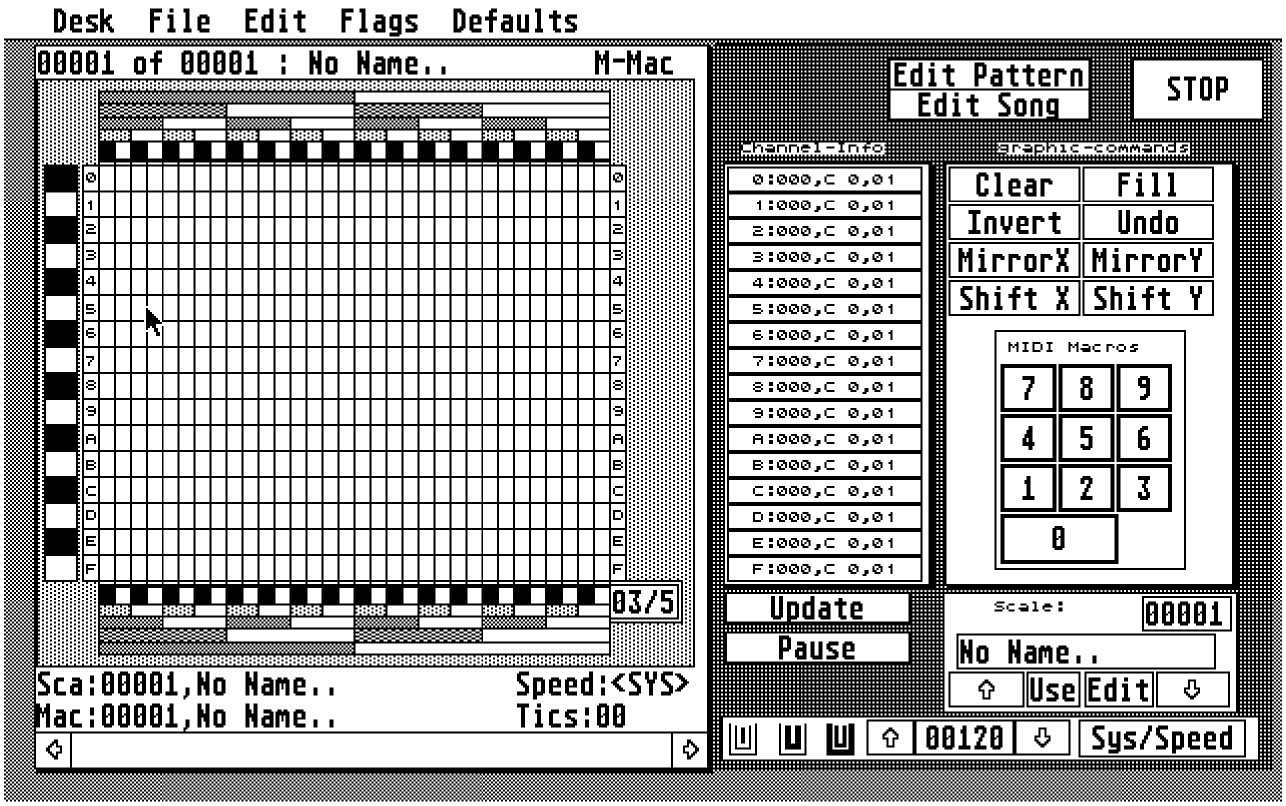
1) Connect your computer' s MIDI Out port to the MIDI In of your keyboard.

2) Connect your computer' s MIDI In port to the MIDI out of your keyboard.

3) For the following experiments your keyboard should be in MIDI Omni Mode (if this is not available receive and transmit channels should be set to MIDI Channel 1).

4) Start '*PAT_C.PRG* (medium resolution) or *PAT_M.PRG* (high resolution).

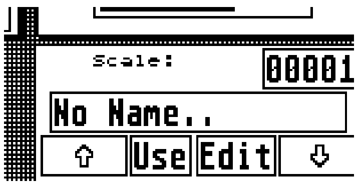
After a few seconds you should be on the **MAIN PAGE** of PATTERNER:



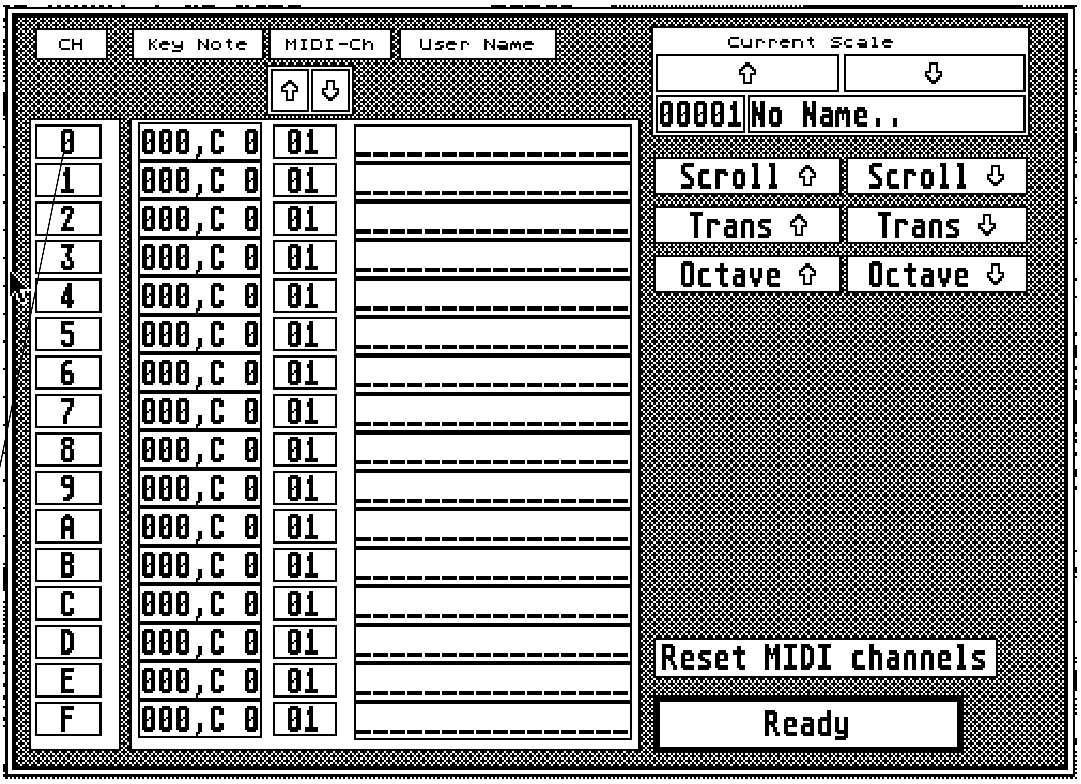
```
00001 of 00001 : No Name..
0:000,C 0,01
1:000,C 0,01
2:000,C 0,01
3:000,C 0,01
4:000,C 0,01
5:000,C 0,01
6:000,C 0,01
7:000,C 0,01
8:000,C 0,01
9:000,C 0,01
A:000,C 0,01
B:000,C 0,01
C:000,C 0,01
D:000,C 0,01
E:000,C 0,01
F:000,C 0,01
Scale: 00001
No Name..
00120
Sca:00001,No Name..      Speed:<SYS>
Mac:00001,No Name..      Tics:00
```

Step 1:

Left-click over the *EDIT* button in the **SCALE/MIDI Macro Switch Box** on the **MAIN PAGE**.



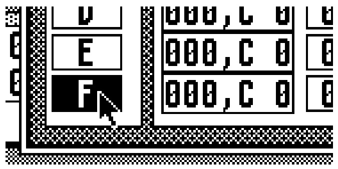
This will open up the **SCALE Editor Box**:



At the very left side of the **SCALE Editor Box** you see a row of sixteen buttons labeled '*0*'-'*F*': The **CHANNEL SELECTION** buttons.

Step 2:

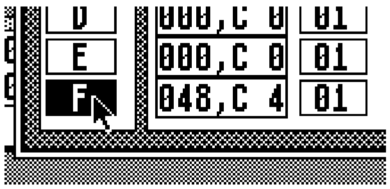
Left-click over the **CHANNEL SELECTION** button with the '*F*' on it. In response the button will turn reverse:



SOUND CHANNEL '*F*' is now active. Whatever key you press on your MIDI keyboard will be sent to this channel.

Step 3:

Push a note '*C*' on your MIDI keyboard. Try different '*C*'s until the **SOUND CHANNEL** displays this one:



```
D  000,C 0  01
E  000,C 0  01
F  048,C 4  01
```

In this picture **SOUND CHANNEL** '*F*' has been set to the note '*C 4*'. In addition to the note's name and octave you see the MIDI Keynumber displayed as decimal value: 48.

Step 4:

De-activate **SOUND CHANNEL** '*F*' by again left-clicking over it. The button returns to normal video.

Step 5:

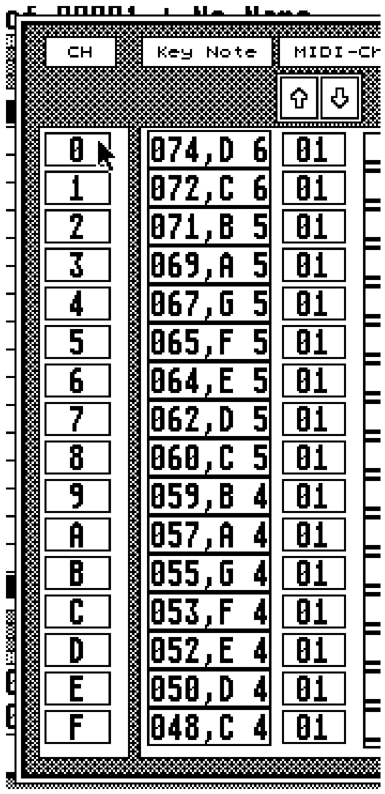
Now, activate **SOUND CHANNEL** '*E*' by left-clicking over that. This **SOUND CHANNEL** is now active and every note that you play on your MIDI keyboard is routed to this channel. Play the note '*D*'. Again, try all the different '*D*'s until you find '*50,D 4*'. De-activate **SOUND CHANNEL** '*E*'.

Step 6:

Repeat Step 5 for all remaining **SOUND CHANNELS** '*D*' to '*0*' setting them to the following notes:

SOUND CHANNEL 'D': 52,E 4
SOUND CHANNEL 'C': 53,F 4
SOUND CHANNEL 'B': 55,G 4
SOUND CHANNEL 'A': 57,A 4
SOUND CHANNEL '9': 59,B 4
SOUND CHANNEL '8': 60,C 5
SOUND CHANNEL '7': 62,D 5
SOUND CHANNEL '6': 64,E 5
SOUND CHANNEL '5': 65,F 5
SOUND CHANNEL '4': 67,G 5
SOUND CHANNEL '3': 69,A 5
SOUND CHANNEL '2': 71,B 5
SOUND CHANNEL '1': 72,C 6
SOUND CHANNEL '0': 74,D 6

After this the **SCALE Editor Box** looks like this:



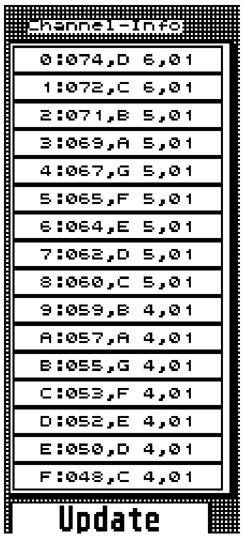
CH	Key Note	MIDI-Ch
0	074,D 6	01
1	072,C 6	01
2	071,B 5	01
3	069,A 5	01
4	067,G 5	01
5	065,F 5	01
6	064,E 5	01
7	062,D 5	01
8	060,C 5	01
9	059,B 4	01
A	057,A 4	01
B	055,G 4	01
C	053,F 4	01
D	052,E 4	01
E	050,D 4	01
F	048,C 4	01

SOUND CHANNELS '*F*' to '*0*' now contain a C-Major **SCALE** from '*C 4*' to '*D 6*'.

Step 7:

Left-click the *READY* button to exit the **SCALE Editor Box** and to return to the **MAIN PAGE**.

The **MAIN PAGE** will look just the same like the opening screen with the exception of the **Channel-Info**:

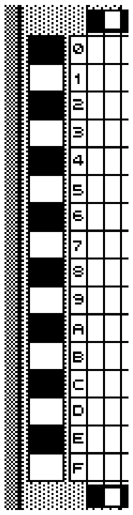


```
Channel-Info
0:074,D 6,01
1:072,C 6,01
2:071,B 5,01
3:069,A 5,01
4:067,G 5,01
5:065,F 5,01
6:064,E 5,01
7:062,D 5,01
8:060,C 5,01
9:059,B 4,01
A:057,A 4,01
B:055,G 4,01
C:053,F 4,01
D:052,E 4,01
E:050,D 4,01
F:048,C 4,01
Update
```

In this **Channel-Info** you can see all the notes you just entered even spelled out in the same way. Each note is displayed to the right of its corresponding **SOUND CHANNEL** in the **BEAT GRID**.

Step 8:

To verify that the notes are really sounding like they are supposed to, do some left-clicking over the buttons of the **Manual Playback Bar** to the left side of the **GRID EDITOR**.

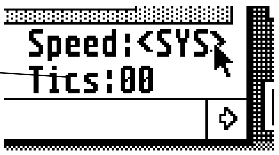


Left-clicking over any of these squares sends the according MIDI Note On message to the keyboard. (This is only a Note On! If you use sounds which don' t fade by themselves after you release the mouse button, hit<*M*> on your computer keyboard to send a '*MIDI Shut up*')

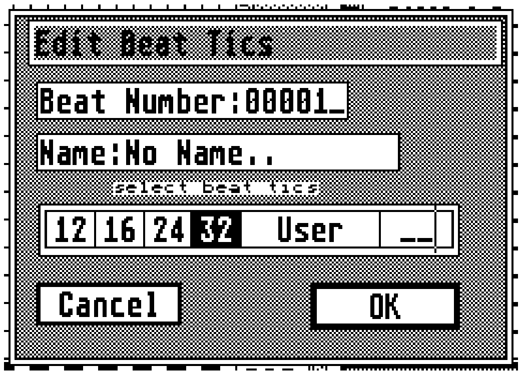
At this point you can playback the notes of your **SCALE** manually but not much more. To use the **GRID EDITOR** follow these next steps:

Step 9:

Left-click over the **TICs** display in the **BEAT GRID EDITOR**



Now you find yourself in the **TICs Editor Box**. Here you can change the name of the current **BEAT** (nothing of interest now) and tell the **BEAT** how many **TICs** it will play back.


Edit Beat Tics

Beat Number:00001_

Name:No Name..

select beat tics

12 | 16 | 24 | 32 | User | __

Step 10:

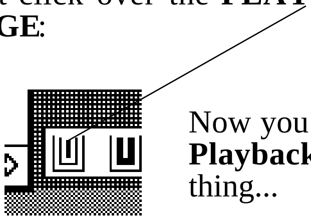
Left-click over the button with the number '*32*' next to the *USER* button. After the '*32*' button has turned reverse exit the box by left-clicking *OK*.

The current **BEAT**'s **TICs** value will now be set to 32 and all of its **TICs** will be played back starting with **TIC** 1, ending with 32.

Now's the time!

Step 11:

Left-click over the **PLAYBACK BEAT** button icon in the **Playback Menu** on the **MAIN PAGE**:



Now you see a message: '*playing BEAT no name...*'. Also you see the flashing **Playback Signal** in the right upper screen corner. But you don't hear anything...

Step 12:

Look at the **GRID EDITOR**. What do you see?

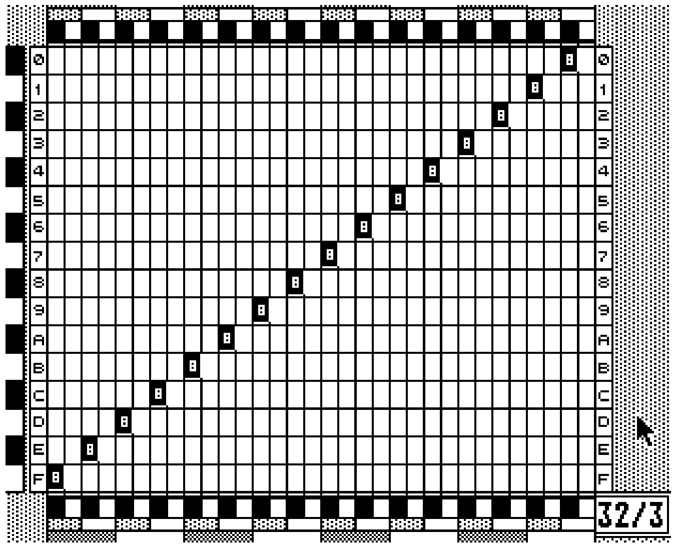
Empty squares!

Right! Now you know why you don't hear anything. Left-click over some squares within the **GRID EDITOR** to fill them. Ahh...

Step 13:

Clear filled squares by left-clicking.

Step 14:



Using the mentioned filling/clearing technique fill the squares shown in this picture:

This finally plays back all the notes of our **SCALE** in an orderly manner. It sounds like a scale excercise.

Step 15:

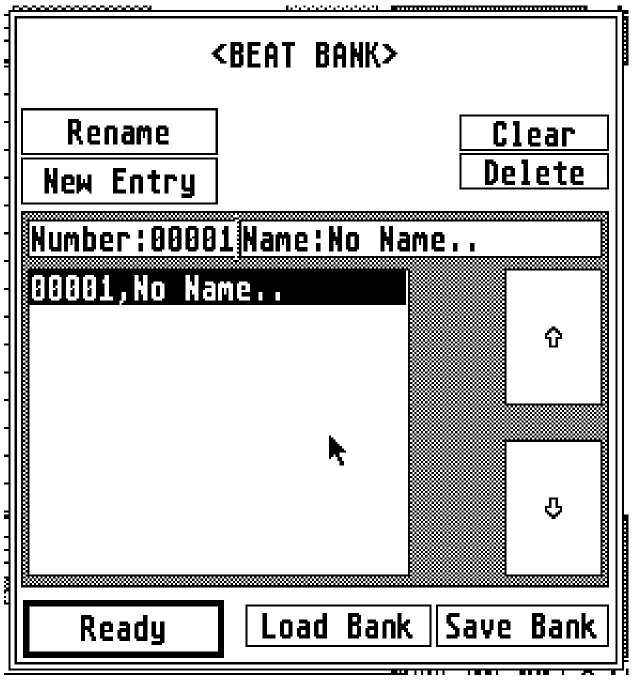
Draw some other patterns and listen to how they sound!

Step 16:

Ok! Let' s name the structures we have just created. You don' t even have to stop playback for this!

Move the mouse pointer over the *FILE* menubar entry. Select the *EDIT* option for **BEAT**.

Now you find yourself in the **BEAT BANK Editor Box**:



Number:00001 Name:No Name..

00001,No Name..

In this box you can add, delete and rename single entries. Or save, load and clear complete **BANKS**.

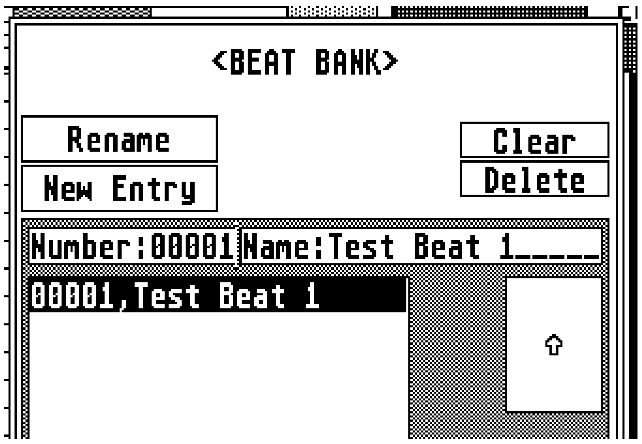
Step 17:

Move the edit cursor (the thin vertical line at the left side of '*Number:00001*') into the '*Name:No Name...*' -field by either left-clicking over the field or using the ST'<*CRSR*> keys.

Press <*ESC*> to empty the name field. Editing this text field is like working in any GEM text field.

Enter yor new name (and DON'T push the <*RETURN*> button!).

Left-click over the *RENAME* button if your name is complete. Now the **BEAT BANK Editor Box** looks like this:



Number:00001 Name:Test Beat 1_____

00001,Test Beat 1

You can see that the **BEAT** you have been working with all the time is the first and only entry in the **BEAT BANK**.

Step 18:

Exit the **BEAT BANK Editor** by left-clicking over the *READY* button.

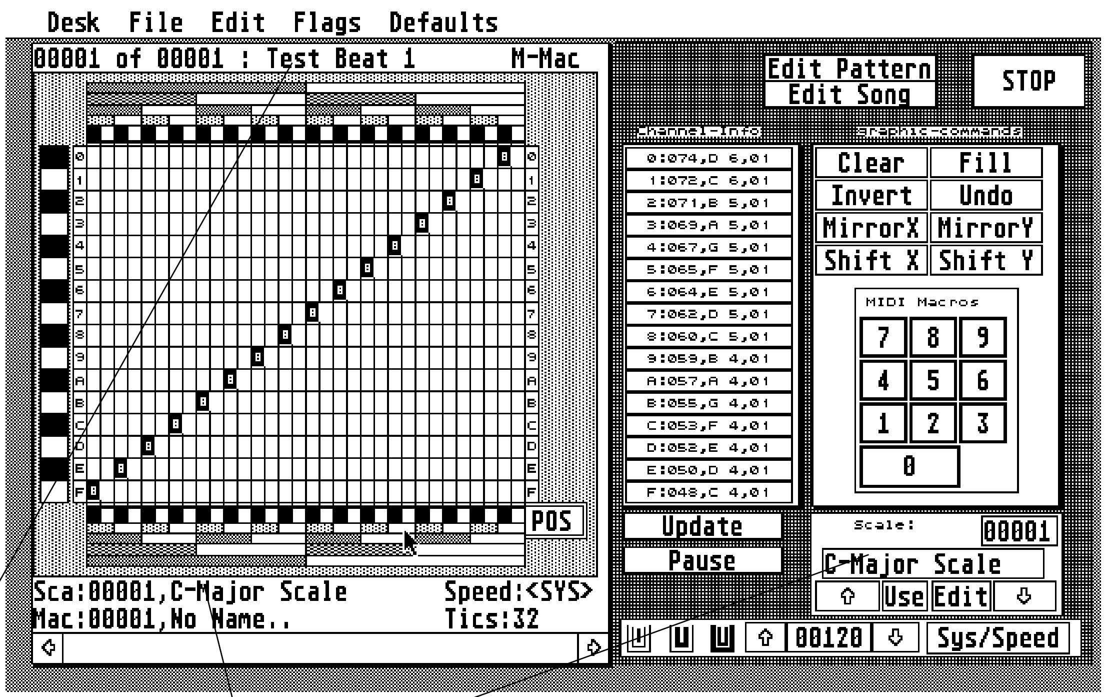
Step 19:

Call up the **SCALE BANK Editor** from the *FILE* menu using the *EDIT* option under **SCALE**.

Change the name of the current **SCALE** the same way you changed the **BEAT** s name in Step 17.

Exit the **SCALE BANK Editor** by left-clicking over the *READY* button.

Now your **MAIN PAGE** should look like this one:



```
00001 of 00001 : Test Beat 1
Sca:00001,C-Major Scale      Speed:<SYS>
Mac:00001,No Name..          Tics:32

0:074,D 6,01
1:072,C 6,01
2:071,B 5,01
3:069,A 5,01
4:067,G 5,01
5:065,F 5,01
6:064,E 5,01
7:062,D 5,01
8:060,C 5,01
9:059,B 4,01
A:057,A 4,01
B:055,G 4,01
C:053,F 4,01
D:052,E 4,01
E:050,D 4,01
F:048,C 4,01

Scale: 00001
C-Major Scale
00120
```

1) You see that the name of your **BEAT** appears just above the **GRID EDITOR**.

2) The name of the **SCALE** appears twice:

a) Just below the **GRID EDITOR**. This line tells you which of all available **SCALES** in the **SCALE BANK** is assigned to the current **BEAT**. In our case we just have this one available **SCALE**.

b) Another **SCALE** name appears in the **SCALE/MIDI Macro Switch Box** in the left lower corner of the **MAIN PAGE**. The contents of this **SCALE** are displayed in the **Channel-Info**.

The two **SCALE** displayes don' t necessarily have to be the same all the time, although they most often are.

Save your work:

In order to preserve what you just created you should save both structures -**BEAT BANK** and **SCALE BANK**- on disk.

To save the **BEAT BANK** select the *SAVE BEAT* option in the *FILE* menu. Select the file path and enter the filename like in any regular GEM Item Selector. You should use a *.BEA* extension for the **BEAT BANK**.

To save the **SCALE BANK** select *SAVE SCALE* in the *FILE* menu. Again, set the path and filename of your choice. Be sure to use a *.ASS* file extension for the **SCALE BANK**.

Up to this point the playback was still running. However, saving files pauses playback and resumes after successfull operation.

Step 20:

At this point you should left-click over the *STOP* button on the **MAIN PAGE** or hit the <*ESC*> key on the computer to stop playback.

Suggestions:

Fool around with the functions you just learnt. Change names, **SPEED**, **TICS** and **s**.

And then we can continue...

TUTORIAL 2

Preparations:

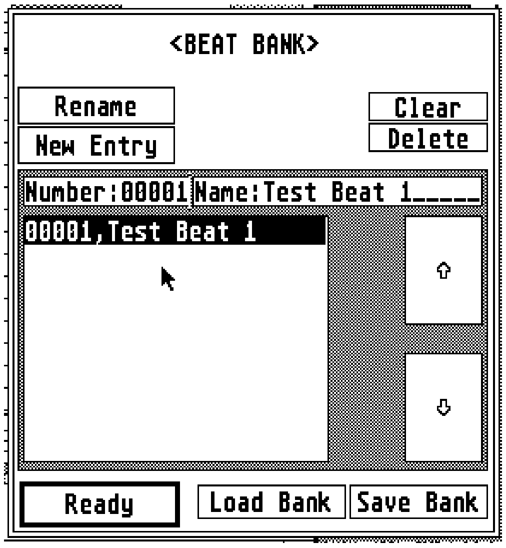
1) Load the **BEAT BANK** we saved at the end of the previous Tutorial 1.

2) Load the **SCALE BANK** we saved at the end of the previous Tutorial 1.

In this Tutorial we will add a couple of entries to both **BEAT** and **SCALE BANKs**.

Step 1:

Using the *BEAT EDIT* option from the *FILE* menu open the **BEAT BANK Edior Box**.



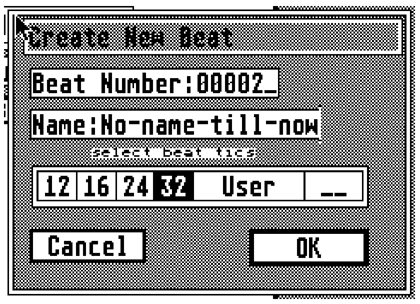
<BEAT BANK>

Number:00001 Name:Test Beat 1_____

00001,Test Beat 1

Step 2:

Click over the *NEW ENTRY* button to bring up the **NEW BEAT Box** which looks just like the **TICs Editor Box** mentioned in Tutorial 1:



Create New Beat

Beat Number:00002_

Name:No-name-till-now

select beat tics

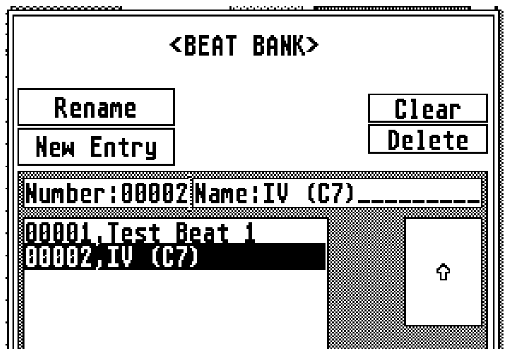
12 16 24 32 User __

In this box you can enter a sixteen character name for your new **BEAT** and also set its **TICs** value.

Push <*ESC*> to clear the Name field and enter '*IV (C7)*' .

Left-click over the '*32*' button to set the **BEAT's TICs** value to 32.

Exit this box by left-clicking over OK.



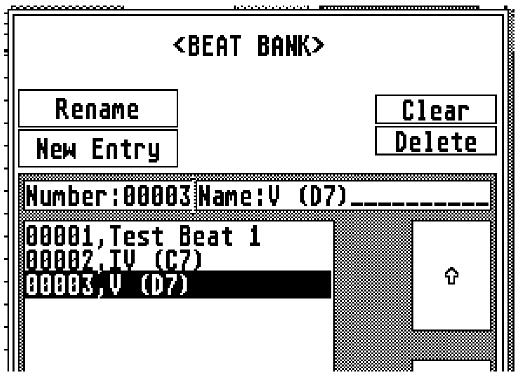
Number:00002 Name:IV (C7)_________

00001,Test Beat 1
00002,IV (C7)

As you can see your new creation appeared in the **BEAT BANK** window.

Step 3:

Repeat Step 2 only this time use '*V (D7)*' as name. Set it to a **TICs** value of 32, again. Left-click *OK* to take over the new data.

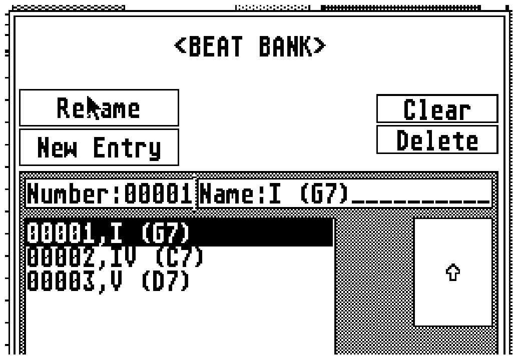


Number:00003 Name:V (D7)__________

00001,Test Beat 1
00002,IV (C7)
00003,V (D7)

Step 4:

Rename the first entry in the **BEAT BANK** which is at the moment called '*Test Beat 1*'. The new name is '*I (G7)*'.



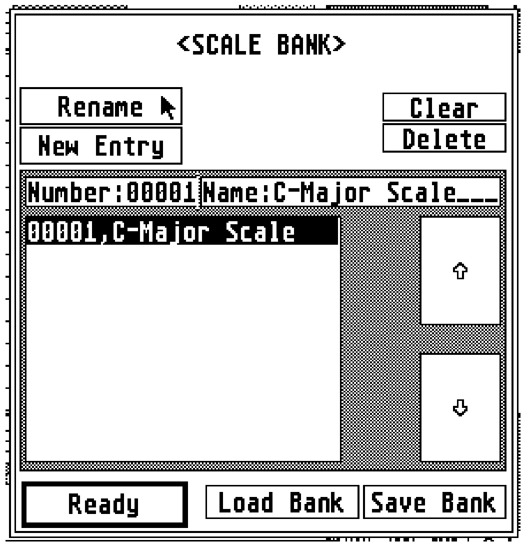
Number:00001 Name:I (G7)__________

00001,I (G7)
00002,IV (C7)
00003,V (D7)

Now we have three entries in the **BEAT BANK**. Let's exit this box by clicking *READY*. You see on the **MAIN PAGE** that the name of the current **BEAT** has changed, everything else, including its contents are the same.

Step 5:

Using *SCALE EDIT* from the *FILE* menu open up the **SCALE BANK Editor Box**.

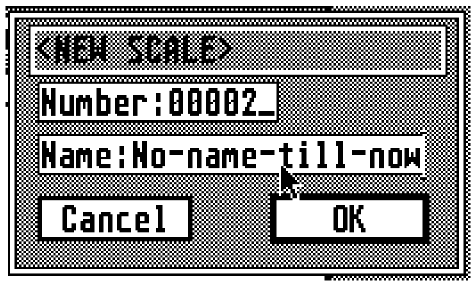


Number:00001 Name:C-Major Scale___
00001,C-Major Scale

Still just one single entry:'*C-Major Scale*' .

Step 6:

Create two new entries in the **SCALE BANK** using the *NEW ENTRY* button and this **NEW SCALE Box**:

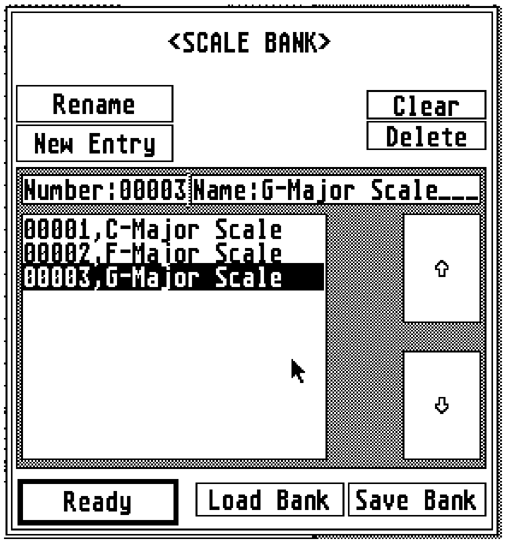


Number:00002_
Name:No-name-till-now

If you compare this box to the **New BEAT Box** you will see that it is even simpler to handle: Just fill in the name of the new **SCALE**.

Step 7:

Our two new entries should be named: '*F-Major Scale*' and '*G-Major Scale*'. Here's the **SCALE BANK** as it looks after all this:


<SCALE BANK>

Rename
New Entry
Clear
Delete

Number:00003 Name:G-Major Scale___

00001,C-Major Scale
00002,F-Major Scale
00003,G-Major Scale

Ready
Load Bank
Save Bank

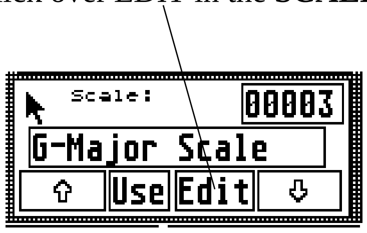
Step 8:

Exit the **SCALE BANK** by left-clicking over *READY*.

You see, that the current **SCALE** display in the right lower screen corner shows '*G-Major Scale*' while the current **BEAT** is still assigned to the '*C-Major Scale*'.

Step 9:

Click over *EDIT* in the **SCALE/MIDI Macro Switch Box** to bring up the **SCALE Editor Box**.


Scale: 00003

G-Major Scale

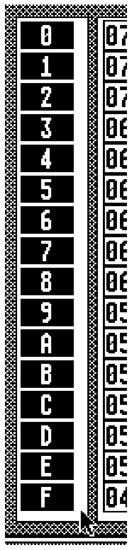
⇧ Use Edit ⇩

As you can see this **SCALE** contains the same MIDI Keynumbers like the '*C-Major Scale*'. The reason for this is that whenever you create a new **SCALE** it will take over the contents of the currently selected **SCALE**.

Step 10:

Since we want a '*G-Major Scale*' in this we obviously have to change something.

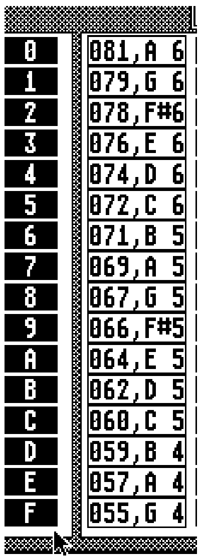
Place the mouse pointer over any one of the sixteen **CHANNEL SELECTOR** buttons. Press the <*ALTERNATE*> key on your ST and keep it pressed. Then left-click! All **CHANNEL SELECTORs** should be selected after this (If not simply try again).



At this point every **SOUND CHANNEL** is selected and will react to incoming MIDI data (don't touch your MIDI keyboard now!) or *EDIT* Commands.

Step 11:

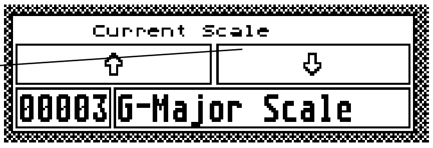
Left-click over the *TRANSPOSE UP* button. As you can see all the Keynumbers are going up a halfstep. Click again and again....until **SOUND CHANNEL** '*F*' displays '*055, G 4*'. Here's our '*G-Major Scale*'.



0	081,A 6
1	079,G 6
2	078,F#6
3	076,E 6
4	074,D 6
5	072,C 6
6	071,B 5
7	069,A 5
8	067,G 5
9	066,F#5
A	064,E 5
B	062,D 5
C	060,C 5
D	059,B 4
E	057,A 4
F	055,G 4

Step 12:

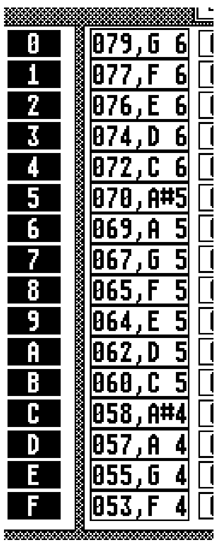
Click over the *DOWN* arrow in the **SCALE Switch Box**:



Now the **SCALE Editor Box** displays the contents of the '*F-Major Scale*'. These contents are still the copy from '*C-Major Scale*' .

Step 13:

The **CHANNEL SELECTORS** should still be selected. *TRANSPOSE UP* until **SOUND CHANNEL** '*F*' displays: '*053, F4*' . Here we got our '*F-Major Scale*' :



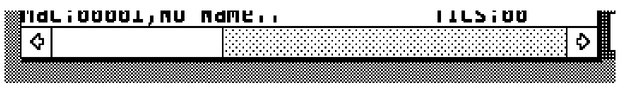
0	079,G 6
1	077,F 6
2	076,E 6
3	074,D 6
4	072,C 6
5	070,A#5
6	069,A 5
7	067,G 5
8	065,F 5
9	064,E 5
A	062,D 5
B	060,C 5
C	058,A#4
D	057,A 4
E	055,G 4
F	053,F 4

Step 14:

We don't have to change '*C-Major Scale*' since those contents are already right. So let's exit the **SCALE Editor Box**.

Step 15:

Back on the **MAIN PAGE**. Use the scroll bar arrows at the bottom of the **EDIT GRID** to browse through the **BEATs** we have.



Look closely at every **BEAT** and you'll see that they are alike except their names at the top of the **EDIT GRID**. They also all use the same **SCALE**:'*C-Major Scale*' .

Step 16:

Our next goal will be to have:

BEAT '*I (G7)*' use **SCALE** '*C-Major Scale*' ,

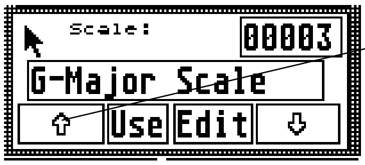
BEAT '*IV (C7)*' use **SCALE** '*F-Major Scale*' and

BEAT '*V (D7)*' use **SCALE** '*G-Major Scale*' .

Use the scroll bar to switch to **BEAT** '*I (G7)*' . As we can see it already uses the '*C-Major Scale*' .

Step 17:

Use the scroll bar to switch on to the '*IV (C7)*' **BEAT**. This uses '*C-Major Scale*' at the moment. So move the mouse pointer to the **SCALE/MIDI Macro Switch Box** and left-click over the *UP ARROW* button.



This brings up '*F-Major Scale*' as current **SCALE** and that's just what we want.

Step 18:

Left-click over the *USE* button to assign the current **BEAT** to the current **SCALE**.

Step 19:

In the same way use the scroll bar to go to **BEAT** '*V (D7)*' and assign it to the '*G-Major Scale*' .

Step 20:

Now start **BEAT** playback by pressing <B> on the keyboard or clicking the **BEAT** playback button in the **Playback Menu**.

Use the scroll bar to scroll through the three **BEATS** and hear them playing back their assigned **SCALEs**.

Save your work:

Like in Tutorial 1 use the **BEAT** and **SCALE SAVE** options from the *FILE* menu to save your **BANKS** to disk. Be sure to take this step now, because we will use these **BANKS** in the next Tutorial section.

TUTORIAL 3

Preparations:

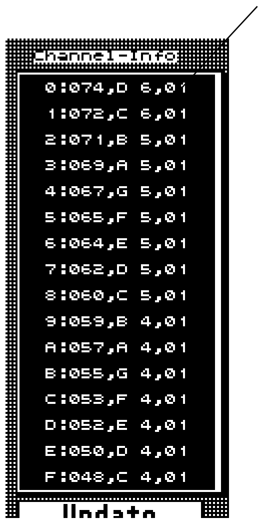
1) Load the **BEAT BANK** we saved at the end of the previous Tutorial 2.

2) Load the **SCALE BANK** we saved at the end of the previous Tutorial 2.

In this Tutorial we will use our already existing **BEATs** and **SCALEs** to assemble a simple 12 bar bluesform. While doing this we will learn a little about some **GRAPHIC COMMANDS**, **PATTERNs** and the **SAVE/LOAD SYSTEM** feature.

Step 1:

Left-click over any of the **MAIN PAGE CHANNEL SELECTORs** in the **Channel-Info** while holding down the <*ALTERNATE*> key on the ST keyboard.

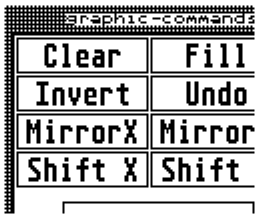


```
0:074,D 6,01
1:072,C 6,01
2:071,B 5,01
3:069,A 5,01
4:067,G 5,01
5:065,F 5,01
6:064,E 5,01
7:062,D 5,01
8:060,C 5,01
9:059,B 4,01
A:057,A 4,01
B:055,G 4,01
C:053,F 4,01
D:052,E 4,01
E:050,D 4,01
F:048,C 4,01
```

Now all **SOUND CHANNELS** are activated and will react to all sorts of **GRAPHIC COMMANDS**.

Step 2:

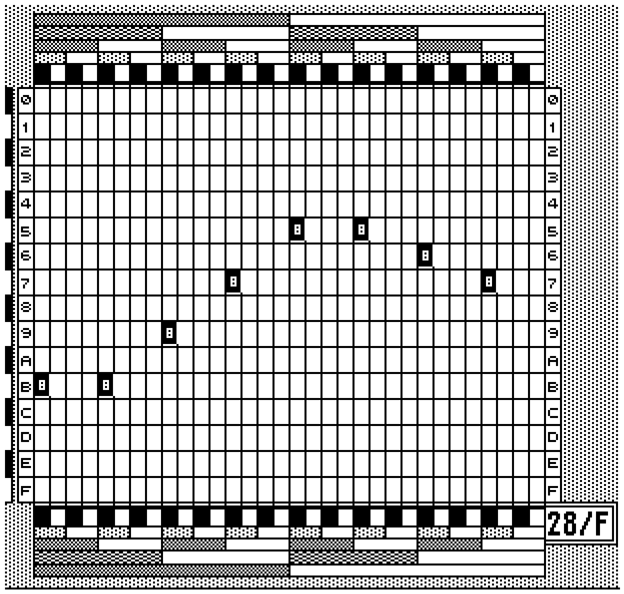
Make sure that the current **BEAT** is '*00001,I (G7)*' then single-left click over the *CLEAR* button in the **Graphic Commands** menu.



As you can see, this has cleared all set positions in the current **BEAT**.

Step 3:

Using the techniques described in Tutorial 1 fill in this pattern of filled and cleared positions:



Step 4:

Push the **BEAT** play button <*B*> or click the respective button in the **Playback Menu**. That sounds like a rock pattern.

Step 5:

Using the **EDIT GRID** scroll bar to switch to the next **BEAT** '*00002,IV (C7)*' . Clear the **BEAT**s contents and fill in the very same filled/clear pattern like in Step 2.

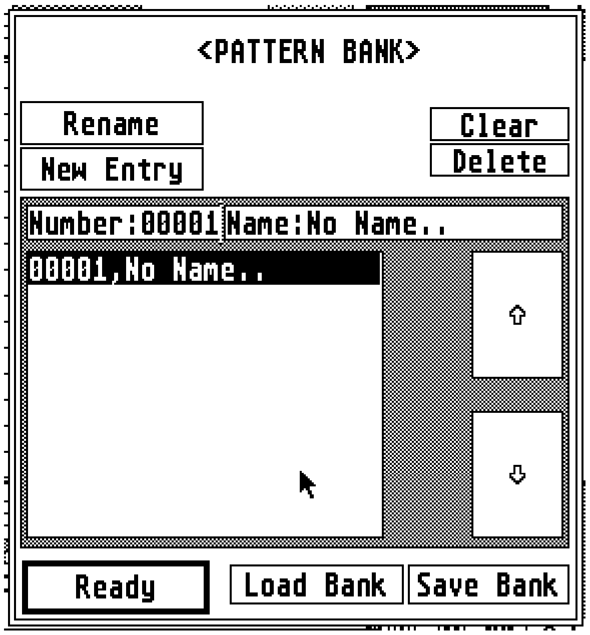
Step 6:

Same as Step 5 only for **BEAT** '*00003,V (D7)*' .

After this we got three identical popular rock rhythms in our **BEATs**. They are in the I, IV and V keys and can be used to build chord structures for a lot of different tunes.

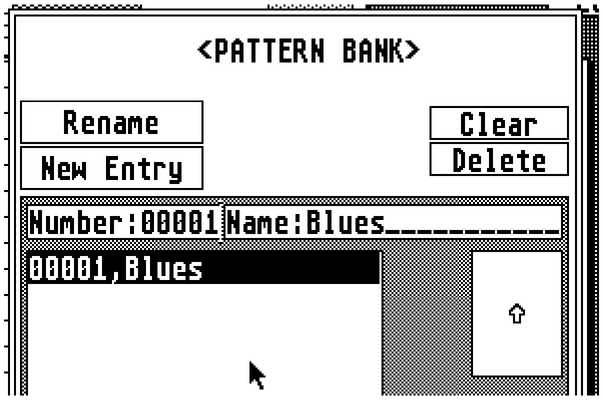
Step 7:

Open up the **PATTERN BANK Editor Box**using *PATTERN EDIT*from the *FILE* menu.



Step 8:

Using the techniques from Tutorial 2 change the name of the only entry into '*Blues*' .

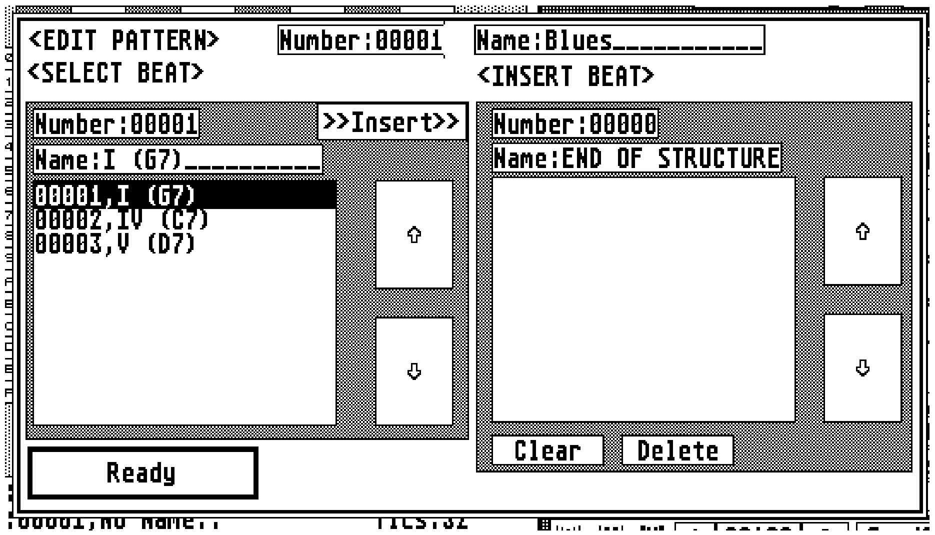


Step 9:

Exit this box by left-clicking over the *READY* button.

Step 10:

Left-click over the *EDIT PATTERN* button on the **MAIN PAGE** to bring up the **PATTERN Editor Box**.



Number:00001 Name:Blues

Number:00001
Name:I (G7)
00001,I (G7)
00002,IV (C7)
00003,V (D7)

Number:00000
Name:END OF STRUCTURE

In this box we can call up all available *PATTERNs* from the **PATTERN BANK** and edit their contents which consist of **BEAT** playback lists.

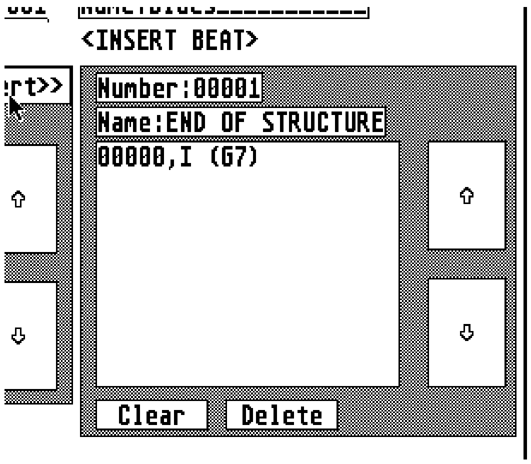
Step 11:

Move the mouse pointer into the left window. This is called the source window and in this case represents a portion of the **BEAT BANK**.

Step 12:

Left-click over the *>>INSERT>>* button in the source window.

This inserts the selected **BEAT** into the destination window which is the current **PATTERN**



Number:00001
Name:END OF STRUCTURE
00000,I (G7)

Note that the name of the inserted **BEAT** is taken over. The number, however, just represents its relative position within the **PATTERN**

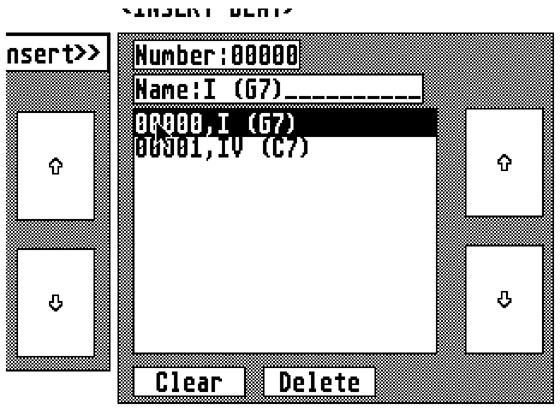
Look at the '*END OF STRUCTURE*' string in the name field above the window. This field contains the insert pointer position which in this case is behind the last entry.

Step 13:

Next, double left-click over our second **BEAT BANK** entry:'*00002,IV (C7)*' . As you can see double-clicking has the same function like >>*INSERT*>>ing an entry.

Step 14:

Let' s fool around a bit. In the destination window single-click over entry'*00000,I (G7)*' . Now the insert pointer is right in front of the selected entry.



Just click over the >>*INSERT*>> button to see what happens.

Aha, the insertion takes place right in front of the pointer position while all the follow-ing entries move one notch.

Step 15:

Since this is not what our **PATTERN**'*Blues*' is supposed to look like let' s get rid of the first des-tination:'*00000,<name>*' . To do this single left-click over its entry in the destination window. As soon as it is selected click over the *DELETE* button below the window and the selected entry will disappear again.

Step 16:

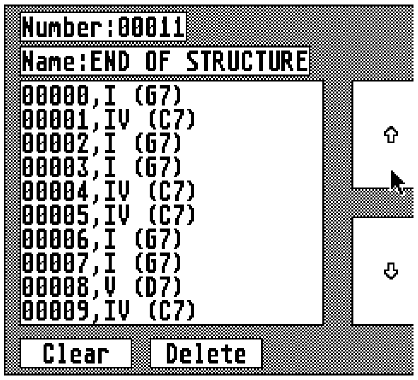
In the destination window single-left click below the last entry. The '*END OF STRUCTURE*' info should once again show up in the destination name field. Now the insert pointer is behind the last entry.

Step 17:

Using the above described techniques of *>>INSERT>>*ing and *DELETE*ing assemble this **PATTERN**:

00000,I (G7)
00001,IV (C7)
00002,I (G7)
00003,I (G7)
00004,IV (C7)
00005,IV (C7)
00006,I (G7)
00007,I (G7)
00008,V (D7)
00009,IV (C7)
00010,I (G7)
00011,V (D7)

Since the destination window only shows you ten entries at one time you will probably have to use the arrow up/down buttons a bit.



```
Number:00011
Name:END OF STRUCTURE
00000,I (G7)
00001,IV (C7)
00002,I (G7)
00003,I (G7)
00004,IV (C7)
00005,IV (C7)
00006,I (G7)
00007,I (G7)
00008,V (D7)
00009,IV (C7)
```

Step 18:

Exit the **PATTERN Editor Box** by clicking over the *READY* button.

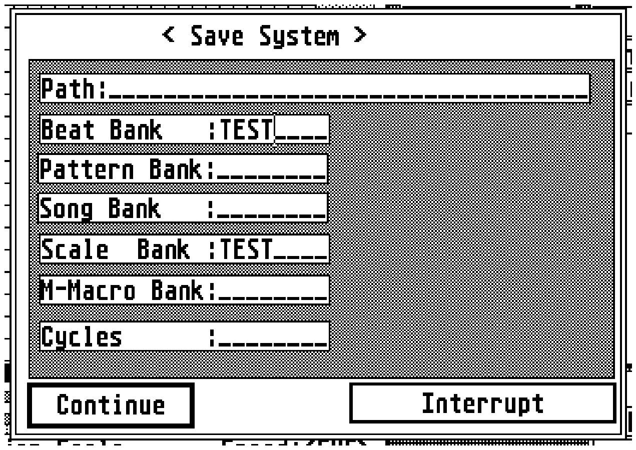
Step 19:

Back on the **MAIN PAGE** either hit <*P*> on the ST keyboard or left-click over the **PATTERN** playback button in the **Playback Menu**:

This plays the complete bluesform for you until you hit <*ESC*> or click over the *STOP* button.

Step 20:

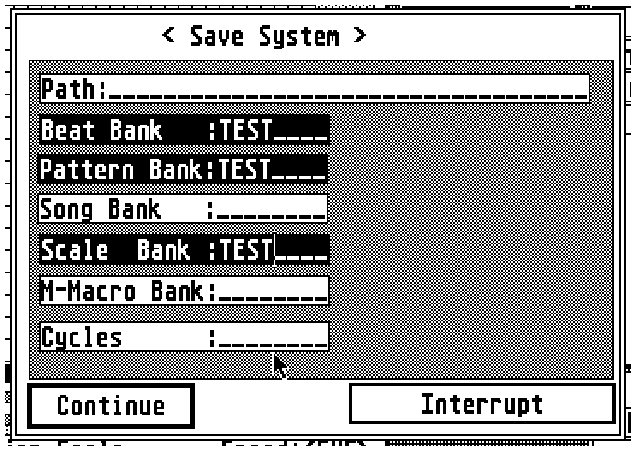
Now it' s time again to save our work. But you may already have complained about having to save two files. Saving three files is out of the question. Select *SYSTEM SAVE* from the *FILE* menu:



In this **SYSTEM SAVE Box** you can select groups of **BANKS** to save and set their names.

Step 21:

Alter the contents of the box to look like this by left-clicking to select fields and entering text as in any regular GEM text field:



Step 22:

Now click over *CONTINUE* to save all the selected **BANKS**. Enter your **SYSTEM'S** name in the upcoming GEM Item Selector (be sure to use a *.SYS* suffix, though). If you save the **SYSTEM** to a folder all **BANKS** in the **s** are saved to the same folder.

See you in the next and last Tutorial.

APPENDIX A

List of file types (refer also to the file format chapter [6]):

.*SYS*
file containing information about the contents of a **SYSTEM** (combination of **BANKS**, flags and playback SPEED)

.*BEA*
file containing a **BEAT BANK**

.*ASS*
file containing a **SCALE BANK**

.*MMC*
file containing a **MIDI Macro BANK**

.*PAT*
file containing a **PATTERN BANK**

.*SNG*
file containing a **SONG BANK**

.*BUF*
file which has been saved by the *SAVE* buffer option. Can be **BEAT/SCALE/MIDI Macro/PATTERN**or **SONG** data.

.*CYC*
file containing a complete **CYCLER** program created by the **CYCLER Box** (assmbly code)

.*SRC*
file containing **CYCLER** source code. Has to be used to make **CYCLER** programs editable after assembly

.*LIB*
file containing a **CYCLER** source library. Assembly code.

(.*Q* - Library assembler source code files (ASCII). Extension may differ with different assemblers. At least the source code file on the program disk uses this extension.)

APPENDIX B

List of available single-stroke keyboard commands from PATTERNER **MAINPAGE**.

<CONTROL> refers to keys which have to be pressed while holding down the *<CONTROL>* key.

<ALTERNATE> refers to keys which have to be pressed while holding down the *<ALTERNATE>* key.

Numbers in brackets refer to the chapter numbers and sub-numbers where to find more information about a specific operation.

<F1>
Launch a *SINGLE/LOCAL* or *GLOBAL* **BEAT MICROCOMMAND** from the **MAIN PAGE**. Available during playback [4.5.3.].

<F2>
Launch a *SINGLE/LOCAL* or *GLOBAL* **SCALE MICROCOMMAND** from the **MAIN PAGE**. Available during playback [4.5.4.].

<F3>
Inverts the status of the **TIC CYCLE FLAG** [4.4.4.]. This and the following strokes (*<F3>* to *<F7>*) are only selectable if a *.CYC* file has been loaded successfully.

<F4>
Inverts the status of the **BEAT CYCLE FLAG** [4.4.5.].

<F5>
Inverts the status of the **PATTERN CYCLE FLAG** [4.4.6.].

<F6>
Inverts the status of the **SONG CYCLE FLAG** [4.4.7.].

<F7>
Inverts the status of the **MANUAL CYCLE FLAG** [4.4.8.].

<1>
Launches **MANUAL** Cycle 1 **[CYCLER** section].

<2>
Launches **MANUAL** Cycle 2 **[CYCLER** section].

<3>
Launches **MANUAL** Cycle 3 **[CYCLER** section].

<4>
Launches **MANUAL** Cycle 4 **[CYCLER** section].

Numberpad <*0*> - <*9*>
Launch **MIDI Macros** as assigned to keyboard in **MIDI Macro Key Box** [4.5.2.].

<*ESC*>
This key stops playback. (same function like *STOP* button on **MAIN PAGE** [3.1.5.]).

<*B*>
Start playback of the current **BEAT** (also from playback menu [3.5.]).

<*P*>
Start playback of the current **PATTERN**(also from playback menu [3.5.]).

<*S*>
Start playback of the current **SONG** (also from playback menu [3.5.]).

<*O*>
Toggle between MIDI- and Internal Data Output (also from *DEFAULTS* menu [4.5.6.])

<*V*>
Open the **VELOCITY Box** [4.5.1.].

<*K*>
Open the **MIDI Macro Key Box** [4.5.2.].

<*F*>
Open the **MIDI IN Filter Box** [4.5.6.].

<*M*>
MIDI Shut Up! Send MIDI Reset/All Notes off [4.5.5.].

<*INSERT*>
Toggle the individual **BEAT MIDI Macro enable flag** (displayed behind the **BEAT** name above **EDIT GRID** [2.1.]).

<HELP>
Updates the current **MAIN PAGE** according to the contents of the playback access array [6.11.]. Same function like *UPDATE* on the **MAIN PAGE** [3.6.].

<CONTROL A>
Open **SCALE BANK EDITOR** [4.2.5.]

<CONTROL B>
Open BEAT BANK EDITOR [4.2.2.]

<CONTROL C>
Open **CYCLE EDITOR** [4.2.7./CYCLER section]

<CONTROL P>
Open **PATTERN BANK Editor** [4.2.3.]

<CONTROL M>
Open **MIDI Macro BANK Editor** [4.2.6.]

<CONTROL S>
Open **SONG BANK Editor** [4.2.4.]

<ALTERNATE A>
Switches between **SCALE Switch Box** and **MIDI Macro Switch Box** on **MAIN PAGE** [3.4.]. (Same like left-click over the *SCALE/MIDI Macro Switch* text string in the box).

<ALTERNATE D>
Toggles between **GRAPHIC COMMAND** page and **extended Channel Info** on **MAIN PAGE** [3.3.]. (Same like left-click over the *Graphic Commands/Channel Info* switch text string).

<ALTERNATE E>
Open the **SCALE EDIT Box** or the **MIDI Macro EDIT Box** according to the display mode of the **SCALE Switch Box/MIDI Macro Switch Box** on the **MAIN PAGE** [3.4.]. (Left-click over the *EDIT* button in the **SCALE/MIDI MacroSwitch Box** has the same effect).

<ALTERNATE P>
Open the **PATTERN BUILD Box** [3.1.3.]. (Same like left-click over *EDIT PATTERN* on **MAIN PAGE**)

<ALTERNATE S>
Open the **SONG BUILX Box** [3.1.4.]. (Same like left-click over *EDIT SONG* on **MAIN PAGE**)

<ALTERNATE T>
Open the **TIC EDIT Box** to change name and number of **TICs** of current **BEAT** [3.1.2.]. (Left-click over **TICs** on **MAIN PAGE** has the same effect).

<ALTERNATE U>
This has the same function like the *USE* button in the **SCALE/MIDI Macro Switch Box** on **MAIN PAGE**. Current **SCALE/MIDI Macro** is taken over into current **BEAT** (according to status of **SCALE/MIDI Macro Switch Box**) [3.4.].

APPENDIX C

List of PATTERNER internal and ATARI Operating System error messages.

APPENDIX C.1

GEMDOS related function errors. Text and error code number:

Invalid Function	-32
File not found	-33
Path not found	-34
No handles left	-35
Access denied	-36
Invalid handle	-37
Invalid handle	-38
Insufficient memory	-39
Invalid memory block address	-40
Insufficient memory	-41
Insufficient memory	-42
Unknown GEMDOS error	-43
Unknown GEMDOS error	-44
Unknown GEMDOS error	-45

Invalid drive specified	-46
Invalid operation	-47
Unknown GEMDOS error	-48
No more files	-49

APPENDIX C2

Internal file handling errors:

Illegal file-format
You tried to load a file which is incompatible to the **BANK** you wanted to load it in (you cannot load a *.BEA* file into the **SCALE BANK** - and if you could you would be in serious trouble).

APPENDIX C3

Edit block errors:

Wrong Block Markers!
If you try to set a *Block End* before a *Block Start* label you' ll get this message.

Edit block definition error!
Traps incorrect *Edit Blocks* before they cause trouble.

APPENDIX C4

CYCLER error messages (at *SAVE CODE* time):

All **CYCLER** error messages in this description make use of the input/output values in the source library you use. If you did write own library routines and tried to use them in **CYCLEs** you have to make sure that your input/output values match in order not to get one of the following errors. There is another case which lets you in fact turn OFF this stack control device. If you do this you will not get any warnings so you have to be sure about what you' re doing.

Not enough values on stack or wrong input/output order in <CYCLE>
This error occurs if one of your **CYCLEs** tries to pick up more values from the stack than there are on it, thus creating an underflow. Another cause might be if you try to pick up a value from the stack before any value has been put there (also causing an underflow).

Stack overflow in <CYCLE>
The internal **CYCLE** stack is limited to 10 data words (values). If your routines write more than 10 values to the stack you get this error message.

Stack input/output unmatch in <CYCLE>
The number of inputs does not match the number of outputs in the shown **CYCLE**.

APPENDIX C.5

BANK relationship errors:

Out-of-range-BEAT error in one or more PATTERN(s)
One or more **BEATs** were deleted which were part of one or more **PATTERNS**. All **PATTERN** entries which tried to access the deletees have been reset to **BEAT** *00001*. This error also occurs when you load a **PATTERN BANK** which contains **PATTERNS** that try to access non-existent **BEATS**.

Out-of-range-PATTERN error in one or more SONG(s)
One or more **PATTERNs** were deleted which were part of one or more **SONGS**. All **SONG** entries which tried to access the deletees have been reset to **PATTERN** *00001*. This also occurs when you load a **SONG BANK** which contains **SONGS** that try to access non-existent **BEATS**.

Out-of-range-SCALE error in one or more BEAT(s)
One or more **SCALEs** were deleted which where assigned to one or more **BEATS**. All **BEATs** which tried to access the deletees have been reset to **SCALE** *00001*. This also happens when you load a **BEAT BANK** which contains **BEATS** that try to access non-existent **SCALES**.

Out-of-range-MMac error in one or more BEAT(s)

One or more MIDI Macros were deleted which where assigned to one or more BEATS. All **BEATS** which tried to access the deletees have been reset to **MIDI Macro** *00001*. This also happens when you load a **BEAT BANK** which contains **BEATS** that try to access non-existent **MIDI Macros**.

APPENDIX D

If you work with the PATTERNER you will be constantly dealing with MIDI data. As long as you don't use **MIDI Macros** or **CYCLES** you won't need to know more about MIDI than the average user.

To use this program's features most efficiently your knowledge about MIDI can't be complete enough. This following list gives you a small overview of several MIDI messages and their format. It makes, however, sense to obtain additional information out of magazines or books dealing with the subject. Also, refer to the MIDI section of your MIDI devices' manuals.

Numbers in this list are given in hexadecimal. (To enter hexdecimal numbers within all PATTERNER number fields use the *'$'* header).

Commands shown are all on basic MIDI Channel 1 (0). I.e. to send a NOTE Off on MIDI channel 3: $82,<Note number>,<Velocity>.

NOTE Off status, bits 4-7

MIDI Channel for this message. MIDI Channels are handled 0-15 internally but 1-16 externally(bits 0-3).

Command	Data 1	Data 2	Description
$80	note number	velocity	NOTE OFF
$90	note number	velocity	NOTE ON
$A0	pitch number	value	POLY PRESSURE
$B0	number	value	CONTINOUS CONTROLLER
$B0	$7A	$00/$7F	LOCAL KEYBOARDS ON/OFF
$B0	$7B	$00	ALL NOTES OFF
$B0	$7C	$00	OMNI OFF
$B0	$7D	$00	OMNI ON
$B0	$7E	$00	MONO ON/POLY OFF
$B0	$7F	$00	POLY ON/MONO OFF
$C0	program number		PROGRAM CHANGE
$D0	value		CHANNEL PRESSURE
$E0	low byte	hi byte	PITCH BEND
$F0	manufacturer ID	data	SYSTEM EXCLUSIVE
$F8			TIMING CLOCK
$FA			SONG START
$FB			SONG CONTINUE
$FC			SONG STOP
$FE			ACTIVE SENSING
$FF			SYSTEM RESET

You will find several of these commands put to work in the **CYCLER's** *SRC* file which comes on the program disk.

APPENDIX E

Glossary

BANK

Most data structures in PATTERNER are organized in **BANKs** (->**BEAT BANK**, ->**PATTERN BANK**..). See [4.2.2.] for more on **BANKs**.

BANK Editor Box

Work within any ->**BANK** is usually done using the **BANK Editor Box** [4.2.2.].

BEAT

A **BEAT** contains timing and velocity information and is displayed on the ->**MAIN PAGE** [2.1.].

BEAT BANK

->**BEATs** are structured in the **BEAT BANK**, where they can be saved and edited [4.2.2.].

BEAT CYCLE

Part of the ->**CYCLE** system which is executed if the respective **BEAT CYCLE Flag** is active [4.4.5. and C.1.2.].

BEAT EDIT GRID

Most of the left side of the ->**MAIN PAGE** containing timing and velocity information [2.1.].

BEAT GRID EDITOR

->**BEAT EDIT GRID**

BEAT MICRO COMMANDS

Group of edit options which can work on one ->**BEAT**, a marked **BLOCK** or all ->**BEATs** in the ->**BEAT BANK** and perform various tasks according to the setting in the ->**BEAT MICRO COMMAND Box** [4.5.3.].

BEAT MMac Flag

Flag used to determine if an individual ->**BEAT** is allowed to launch a ->**MIDI Macro** [2.1. and 2.5.].

BEAT TRIGGER

The first section of a ->**BEAT CYCLE**.

BLOCK BUFFER

PATTERNER's Block Operations use this buffer to store data [4.3.].

BLOCK MARKER Box

In this box you set **BLOCK MARKERS** in the different ->**BANKs**. Marked **BLOCKS** are used for a variety of purposes [4.3. and 4.5.3.].

Channel-Info

Info field almost in the center of the ->**MAIN PAGE** displaying information about the contents of the sixteen ->**SOUND CHANNELs** of the current ->**SCALE** [3.2.].

CHANNEL SELECTION PATTERN
Single-left clicking over any of the ->**CHANNEL INFO** buttons lets you select/de-select ->**SOUND CHANNELs**. The **CHANNEL SELECTION PATTERN**is used for many different purposes [3.2., 4.5.4., **CYCLES**].

Color Velocity
The only main difference between monochrome and color versions of PATTERNER consists of the representation of the velocity of filled ->**GRID POSITIONs** [4.5.1.].

CYCLE
User programmable part of the ->**PLAYBACK** routine [2.7., 4.4., 4.2.7. and **CYCLES**].

CYCLER Box
Used to assemble ->**CYCLEs** from an existing ->**CYCLER LIBRARY** [**CYCLER** References].

CYCLER LIBRARY
Disk file which comes on the program disk containg a number of assembly routines which can be put together to complete ->**CYCLEs** in the ->**CYCLER Box** [**CYCLER** References].

DEFAULT VELOCITY
Used to determine a velocity value which is required for many actions. This value is set in the ->**VELOCITY Box** [3.3., 4.5.1.].

Enable MIDI Macro Flag
Flag to enable or disable the automatic launch of ->**MIDI Macros** on a global level [4.4.]

Global MIDI Macro Flag
->**ENABLE MIDI Macro Flag**

Graphic Edit Commands
A set of buttons to influence the contents of the ->**BEAT EDIT GRID** [3.3.].

GRID POSITION
A position within the ->**BEAT EDIT GRID** which can be filled, cleared or changed in its velocity [2.1.].

GRID WINDOW
Left side of the ->**MAIN PAGE** containg the ->**BEAT EDIT GRID** [2.1.].

INTERNAL SPEED
Every ->**BEAT** can use its own **SPEED** value or the global ->**SYS SPEED** [2.1.].

MAIN PAGE
The screen as it looks right after starting the PATTERNER with no forms opens.

MANUAL CYCLES
Section of the ->**CYCLEs** which can be executed from numberkeys <*1*>-<*4*> on the ST keyboard [2.7., 4.4.8., **CYCLES**].

Menubar
The well known row of menu selections at the top of the ->**MAIN PAGE**.

MICRO COMMANDS

->**BEAT MICRO COMMANDs** and ->**SCALE MICRO COMMANDs**

MIDI IN Filter Box

The ->**CYCLER** gives you access to a MIDI In buffer. The **MIDI In Filter** gives you a little control over incoming data [4.5.7.].

MIDI Macro

Flexible data structure stored in the ->**MIDI Macro BANK** which can contain any MIDI (or not MIDI) data. **MIDI Macros** are created in the ->**MIDI Macro COMPILER** [2.5., 3.4.2.].

MIDI Macro BANK

Holds all ->**MIDI Macros**

MIDI Macro COMPILER

In this you create and edit ->**MIDI Macros** [2.5., 3.4.2.].

MIDI Macro Key Box

->**MIDI Macros** can be called automatically or by pressing keys from the ST' s number keypad. In the **MIDI Macro Key Box** you can edit which key corresponds to which ->**MIDI Macro** [5.2.].

MIDI Macro Send Buttons

A set of ten buttons below the ->**GRAPHIC EDIT COMMANDS** on the ->**MAINPAGE** with identical function like the number keys on the ST number-pad:launch ->**MIDI Macros** manually [3.3.1.1.].

Number/Name Fields

GEM text input fields which are used in many forms in the PATTERNER [1.3., 2.4., 2.7., 4.2.2.].

PATTERN

->**BEATs** can be structured into larger ->**PLAYBACK** units called **PATTERNS** [2.3., 4.2.3.].

PATTERN CYCLE

A part of the ->**CYCLEs** which is used in the ->**PLAYBACK** routine after every completed **PATTERN** (Manual 2.7., 4.4.6., **CYCLER** Booklet 1.3.).

PATTERN Editor Box

Used to assemble ->**BEATs** into **PATTERNs** [3.1.3.].Also called ->**BUILD Box**.

PATTERN TRIGGER

Section of the ->**PATTERN CYCLE** where the playback routine goes after every completed ->**PATTERN**

Playback

...is on whenever the playback sign in the upper right screen corner is flashing. **PLAYBACK** functions are controlled from the Playback menu [3.5.].

SCALE

Contains information about which MIDI Keynumber and MIDI Channel is active for every ->**SOUND CHANNEL** [2.2.].

SCALE BANK
->**SCALEs** are held in the **SCALE BANK** [4.2.5.].

SCALE Editor Box
The contents of ->**SCALES** are edited in the **SCALE Editor Box**.

SCALE MICRO COMMANDS
Tools which are affecting single ->**SCALEs**, a marked block of them or all ->**SCALEs** in the ->**SCALE BANK** and which are defined in the **SCALE MICRO COMMAND Box** [4.5.4.].

SCALE/MIDI Macro Switch Box
Small but important field on the ->**MAIN PAGE** just below the ->G**RAPHIC EDIT COMMAND** field. Used to switch between ->**SCALE** and ->**MIDI Macro** editing [3.4.].

SONG
Frame structure in the PATTERNER. Holds groups of ->**PATTERNs**[2.4., 4.2.4.).

SONG CYCLE
Section of the ->**CYCLEs** which is executed after every finished->**SONG**.

SONG Editor Box
Used to assemble single ->**PATTERNs** into **SONGs** [3.1.4.].

SONG TRIGGER
Section of the ->**CYCLEs** where the playback routine jumps after every finished ->**SONG** when the ->**SONG CYCLE** is active.

SOUND CHANNEL
The ->**BEAT EDIT GRID** is vertically divided into sixteen different **SOUND CHANNELs** each of which gets its information from the attached ->**SCALE** [2.2., 3.2., 2.3.]. **SOUND CHANNELs** should not be mixed up with MIDI Channels.

SYS Speed
The global ->**PLAYBACK** speed of the PATTERNER as displayed at the center of the **Playback menu** [3.5.4.].

SYSTEM
After creating a bunch of ->**BEATs**, ->**PATTERNs**,->**MIDI Macros** and ->**PATTERNs** ...etc. you can save everything as a **SYSTEM** which is able to restore all files like they where and which also sets all flags like they were when the **SYSTEM** file was created [2.6., 4.2.1.].

Text Mode
Used in the ->**SCALE Editor Box** to enter info text about different ->**SOUND CHANNELs** [2.2., 3.3.2.].

TIC
Basic playback unit. One **TIC** represents on vertical row of ->**GRID POSITIONS** in the ->**BEAT EDIT GRID** [2.1., 4.4.4.].

TIC CYCLE
Portion of the ->**CYCLEs** which is executed after every finished ->**TIC** [4.4.4., C. 1.1.].

TICS Edit Box
To edit the ->**TIC** value open this box [2.1.].

TIC TRIGGER
Section of the ->**TIC CYCLE** where the playback routine enters after every finished ->**TIC**.

User Names
...can be entered in the ->**SCALE EDITOR Box** in ->**TEXT MODE** to describe functions of different ->**SOUND CHANNELs** [2.2.].

Value Enter Box
Used in the ->**CYCLER Box** to enter decimal, hexadecimal and binary values [4.2.7. and C.3.1.].

VELOCITY
MIDI volume value. **VELOCITIEs** in PATTERNER are displayed in filled ->**GRID POSITIONs** and can be edited in many ways [2.1., 4.4.3., 4.5.1.].

VELOCITY Box
Used to set ->**DEFAULT VELOCITY** and to alter ->**VELOCITIEs** of specific ->**GRID POSITIONs** [2.1., 4.5.1.].