Dealing With ASCII

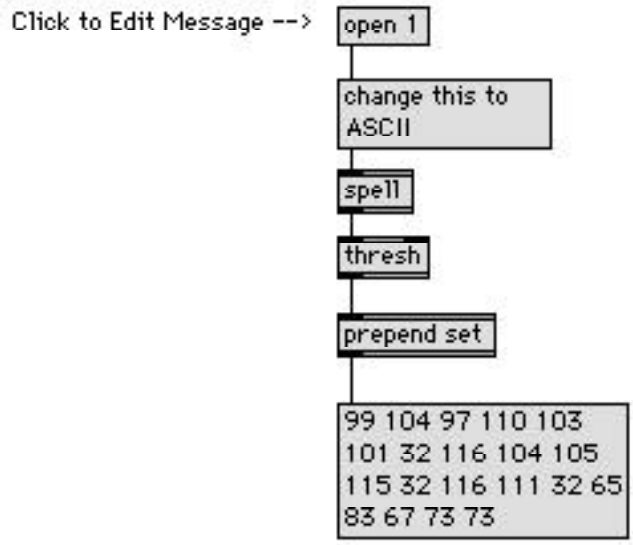
ASCII, of course, is the numeric representation of letters used in most computers. In ASCII, there is a number for each character in a message. Max does not use ACSII very much. In the Max system, one number (called a symbol pointer) is used to represent each word. The program does not know all possible words, it simply assigns a symbol pointer to each word as it is entered. (First it checks to see if the word has already been entered, if it has, the existing symbol pointer is used.) These symbol pointers can be passed around very efficiently; only display objects need to look up the complete ASCII message. The type of message called a symbol is really nothing more than one of these pointers.

This is great when speed is important and relatively few words are necessary, but Max would be a terrible language for writing a text editor.

From time to time we need to convert symbols to ASCII or vice versa.

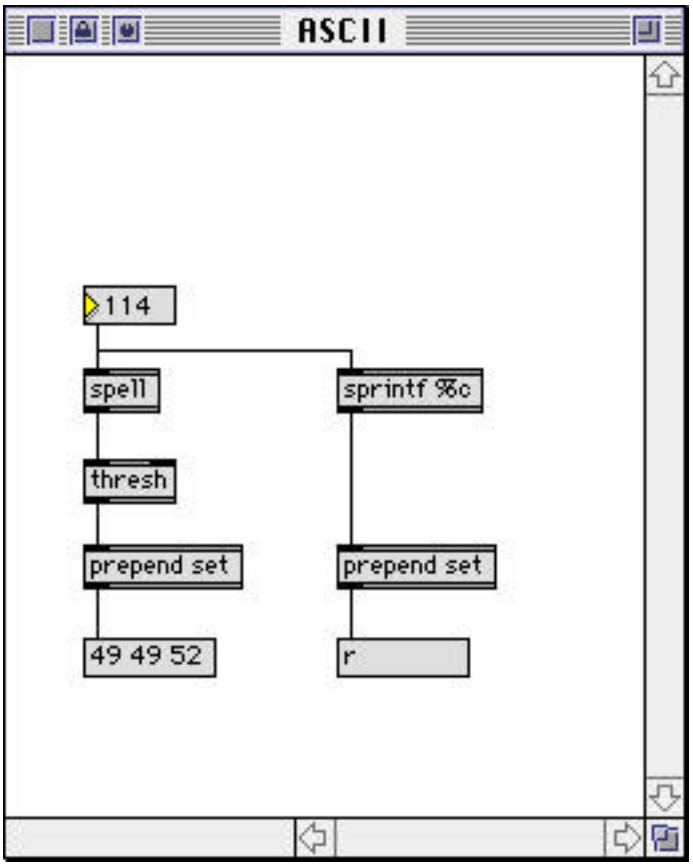
Symbols to ASCII

To get the ASCII out of a symbol, use the spell object. A single symbol is converted to a spurt of ints. If several symbols are input as a list, the ASCII space (32) is inserted between each symbol. If you need to convert a series of individual symbols, you probably need to provide the space yourself. This shows how it works:



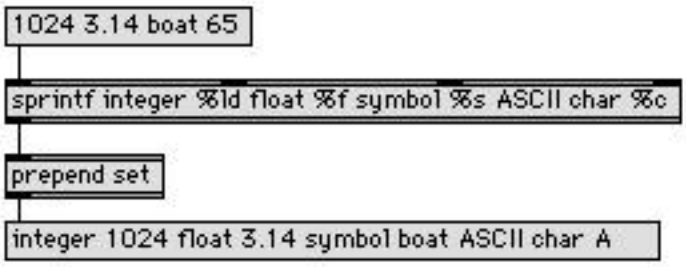
If you study the numbers in the bottom box, you can figure out the ASCII for "ASCII" is 65 83 67 73 73.

If an int is applied to spell, the output is the ASCII representation of the digits of the int. It is important to keep this distinction clear. The number 114 ASCIIizes to 49 49 52. (Here's a quick way to convert a digit to ASCII: add 48.) This patcher illustrates the difference:



ASCII to Symbols

The sprintf object can be used to convert ASCII into symbols. Sprintf is a " string print formatter", in which parts of a message can be replaced by input data. The arguments to sprintf are the words of the message and placeholders %ld %f %s or %c. This example shows what happens.



The help file and manual are not complete of their description of sprintf, referring you to "any standard C language reference", not a common item on most coffee tables. Here's what the placeholders mean:

%ld or % d	Replace with an int and display as decimal. (Floats are truncated.)
%x or % X	Replace with an int and display as hexadecimal.
%o	Replace with an int and display as octal. (There's one you don't see much any more!)
%u	Replace with an int and display unsigned. (In theory, this should let you display numbers larger than 2874836647, but it doesn't.)
%f	Replace with a float. The format %.2f will round the float to 2 decimal places.
%s	Replace with a symbol. The format %.2s will show the first 2 letters of the symbol.
%c	Replace with character.

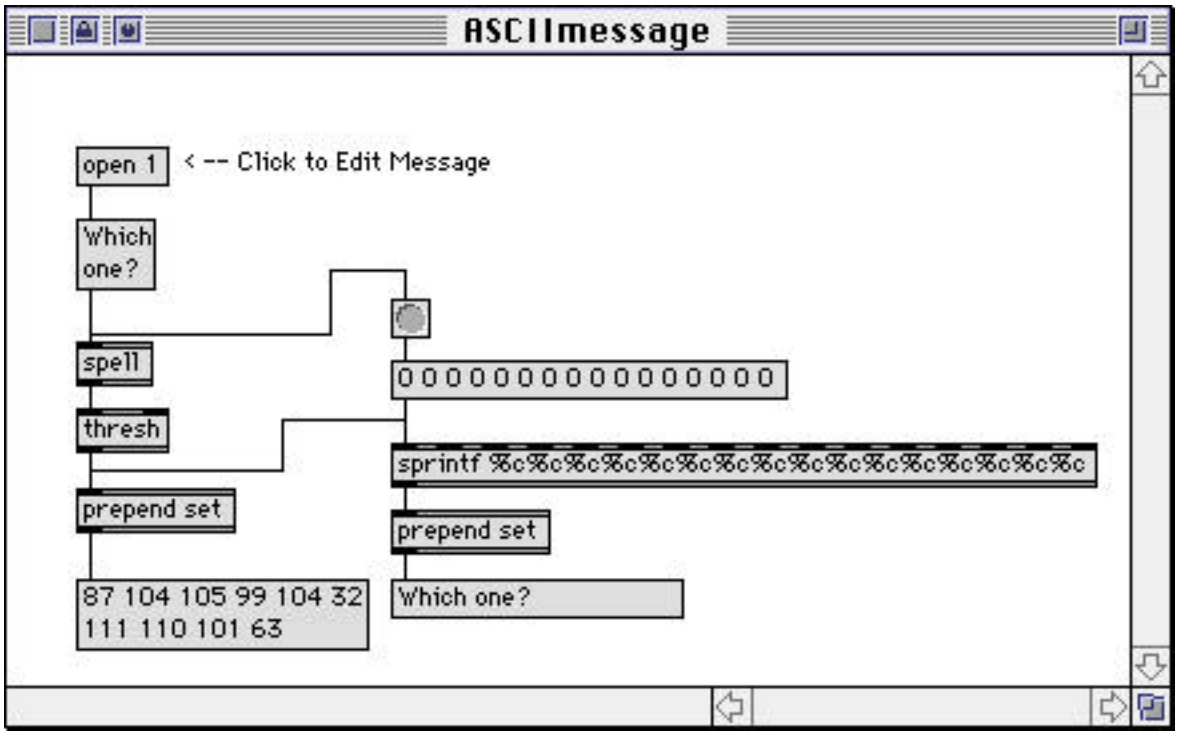
There are other options in the C language printf, but they are not really appropriate to Max and not implemented. (Or partially implemented- try %05.5X) (A warning - C allows %* in some cases- that one will bomb.)

We are interested in the %c placeholder, which converts an ASCII value into the appropriate character. To convert whole words, use this:

Of course, you are limited in the number of %c's you can put in there. (more than 32 will crash some versions of Max.) If the ASCII codes include spaces (32), the output will be a list of distinct symbols. If a group of characters is all digits, a number will be generated instead of a symbol.

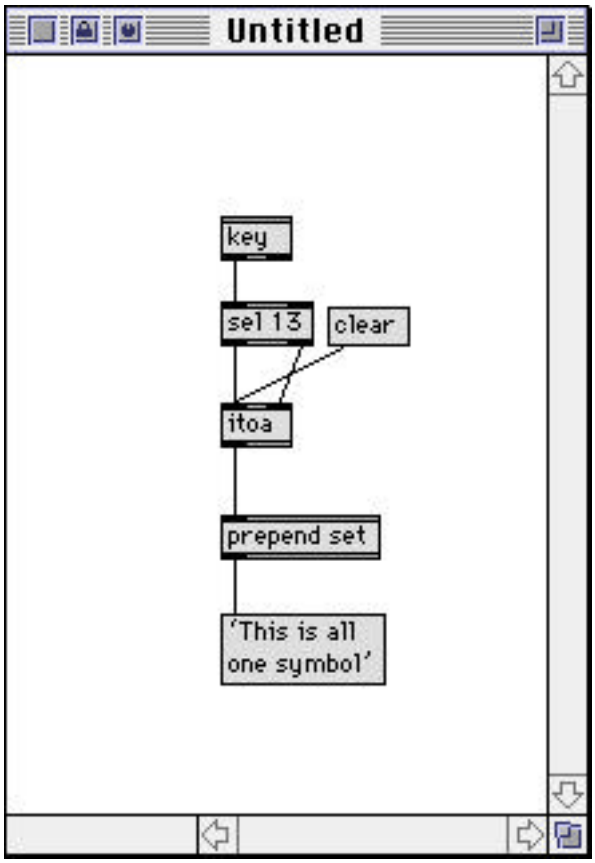
Sprintf works like pack, in that each inlet remembers the last input, so you have to clear it with a list of 0s (non printing characters) between words.

Here is a patcher that illustrates the use of spell and sprintf:



I opened the first message box and typed "which one". Spell converted this into ASCII which thresh made into a list, displayed in the lower left message. Sprintf converted the ASCII list back into a list of symbols. When you do this sort of thing, you'll have to warn the user about reserved words. Try this example with the phrase "bang the drum slowly".

You can also convert ASCII into symbols with the itoa object. This is an "unsupported external" by Steve Ellison. If you apply a list of ints to the left inlet, a symbol is output. At the right inlet, itoa works somewhat like accum. ASCII characters are gathered until there is a bang at the left.

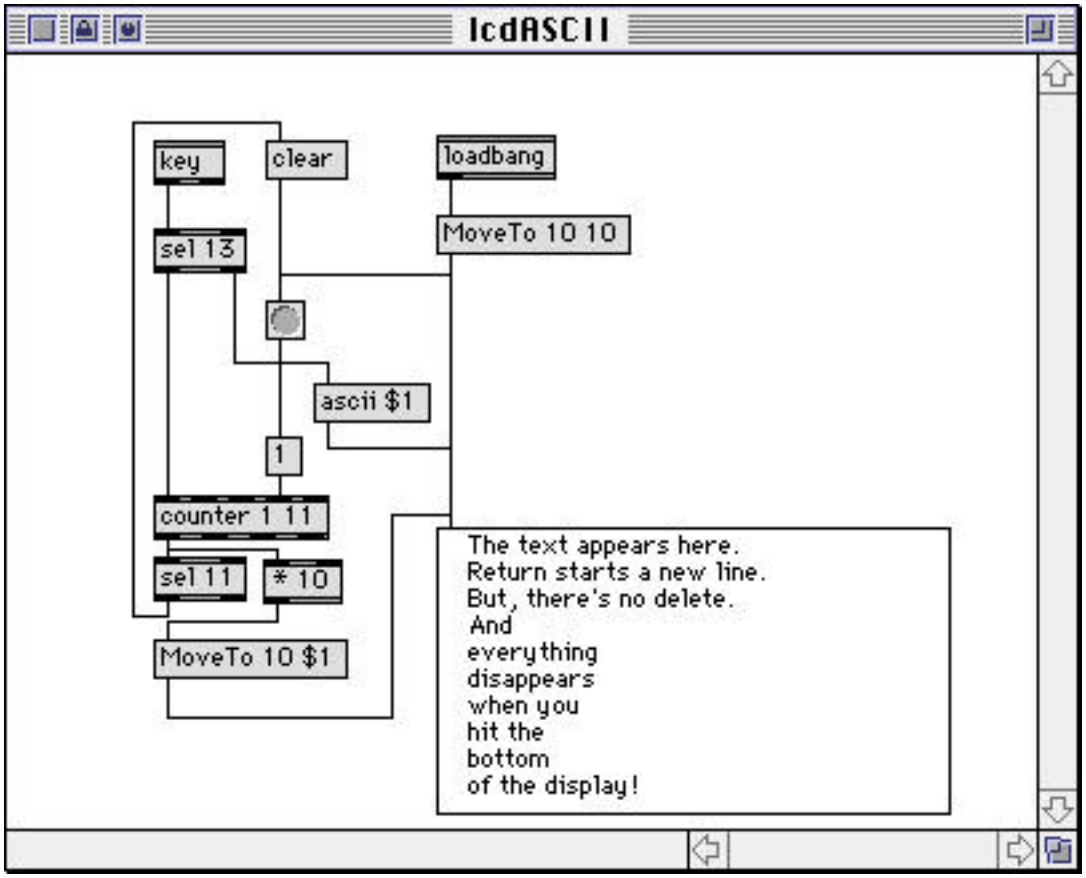


Note that if your symbol contains space characters, message boxes will display it enclosed in single quotes. If you should happen to convert that symbol back to ASCII with spell, the single quotes will not be included.

There is little reason to create new symbols at run time. If you do much of this, the symbol table will eventually grow to unmanageable size, with a noticeable slowdown of all symbol operations. If ASCII is coming in, you should deal with it directly. Here are a few techniques:

Display ASCII Text

To display ASCII, use the lcd object. The lcd is basically a quickdraw graphport; it has a lot of functions, including drawing with the mouse. The ASCII message will put a letter on the screen. To manage text, you need to position the cursor before writing and figure out how to wrap to the next line. This patcher demonstrates a couple of mechanisms:



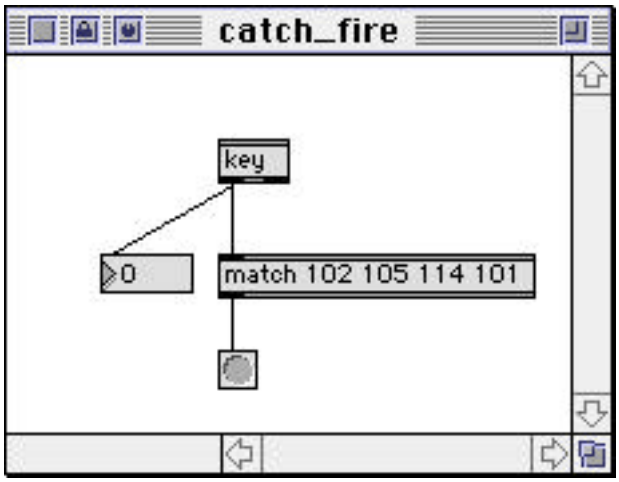
This is very limited. It will do for showing text from a serial port, but I wouldn't want to use it for any data entry. (I need my delete key!)

The ASCII 13 that triggers a move to the next line is a carriage return, as supplied by the Mac return key. This is used by apple to indicate the end of a line. If you are getting serial data from a UNIX type system, end of line will be indicated by ASCII 10, the linefeed or "newline" character. Serial data sent from PCs use both.

Enter, Edit And Save Text

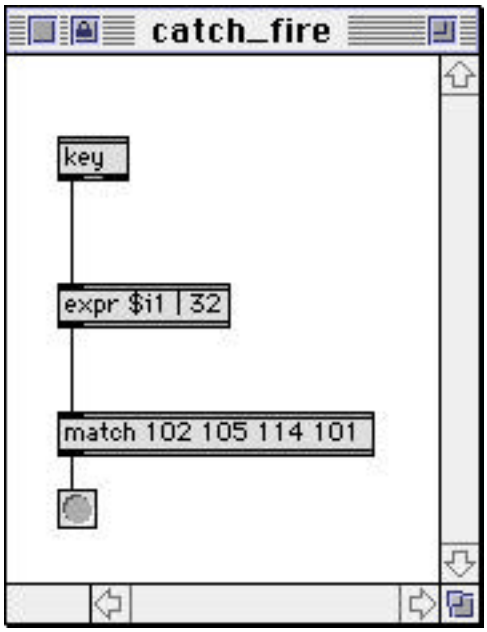
The text object provides entry of complex text. The help file illustrates its capabilities pretty well. Basically, the user can open a window and type in anything he pleases. The contents are converted to symbols when the window is closed, and can be dumped a line at a time. The whole works can be saved as a text file, other files can be read in, and so forth.

Detect And Respond To Typed Commands



All you have to do is figure out the ASCII string you are looking for and watch for it with match. In this case, 102 105 114 101 spells fire. Notice the number box hanging on the key object. That's how I figure out ASCII values. (It's faster than any chart.)

This is case sensitive, which can be a pain. To make caps irrelevant, do this:



The expr $i1 | 32 takes advantage of the fact that the lower case ASCII characters are the same as the upper case with the 6th bit set. This will change some of the punctuation too, so if you are expecting any, use a split object to separate it out. (To find out more about OR (|), read the essay on Max & numbers.)

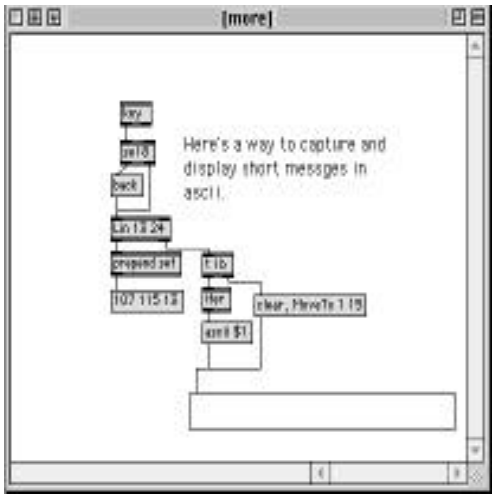
You may find it interesting to use a binary number box to watch the shift and control keys modify the ASCII numbers. The option keys also change the ASCII code from a key, but not in any orderly fashion.

We are getting ASCII out of the left outlet of the key object. Not all of the Mac keys can be read with ASCII, however. (The extra keys return ASCII, but many of them have the same values.) If you want to trigger events off the function keys or the arrow keys, you need to use the right outlet of key, which displays Apple's key codes. Every key on the keyboard has a unique code: F1 is 122, F2 is 120. They don't make any sense, but each key code is unique.

Incidentally, you can't detect the help key by any method, because Max has its own idea of what to do when you press the help key. To define other command-key combinations, use the menubar object.

Using Lin to get ASCII

I've written an Lobject to simplify the input of ASCII from the keyboard. The helpfile pretty much explains it:



The right outlet of Lin will give the message in progress (including deleting the last character if the message "back" is received.) the message will be sent out of the left outlet when a return is input.

You can easily detect expected messages with the Like object.

ASCII Codes

Here's a table of the ASCII codes. The enigmatic ones like ETB apply to teletypes and the like. You might get them from some serial applications. The parentheses indicate the Mac key that generates them.

0	NUL	34	"
1	SOH (Home)	35	#
2	STX	36	$
3	ETX (enter)	37	%
4	EOT (end)	38	&
5	ENQ	39	'
6	ACK	40	(
7	BEL	41	)
8	BS (delete on Mac)	42	*
9	HT (Tab)	43	+
10	Line feed	44	,
11	VT (Page Up)	45	-
12	Form Feed (Page Down)	46	.
13	CR (return)	47	/
14	S0	48-57	numbers 0 -9
15	SI	58	:
16	DLE (all F keys)	59	;
17	DC1	60	<
18	DC2	61	=
19	DC3	62	>
20	DC4	63	?
21	NAK	64	@
22	SYN	65-90	Capitol A -Z
23	ETB	91	[
24	CAN	92	\
25	EM	93	]
26	SUB	94	^
27	escape (also clear)	95	_
28	cursor right	96	`
29	cursor left	97 - 122	letters a - z
30	cursor up	123	{
31	cursor down	124	|
32	space	125	}
33	!	126	~
		127	DEL