

INTRODUCTION TO MAX

What is this thing called Max, and why is PQE so hot on it?

Time was, electronic music was about inventing new ways to do music. It was about new ways for music to sound, new ways to play, new ways to compose. It was exciting, a bit scary, and fun. We could grab anything that seemed capable of making noise and play with it until we found something entertaining about it. Computers were particularly intriguing, because they could do anything! This all gradually changed. Electronic music became an industry, a genre, a market. Computer programs became user friendly, instruments were standardized, and one day I looked around and saw electronic music was as tradition bound and predictable as baroque opera. What's worse, the instruments and programs were musically and technically condescending. We were gradually being limited to safe sounds like imitations of band instruments and one patch on the old Moog. The program interfaces looked like tape recorders and score paper, only you weren't allowed to do things that are easy on real tape recorders and score paper. Yes we can do a lot, and wild new plug-ins and programs are released all the time. But all the new stuff is the same story. Dull music is easy (Everyone can be a star! No music training required!), interesting music is hard and you have to buy a \$500 program for that one little trick it does well. Of course, a few of us aren't limited by the commercial software market, we can write our own code. Problem is, writing programs takes so much time, there's none left to do music.

Max is different. Max is designed to be simple to use, but not by limiting you to simple results. Max is wide open—you can put in exactly the options you want, put the controls where you want them, leave out features you don't want to use. You are making your own programs with Max, you just don't have to spend all that time chasing errant semicolons and rewriting things other folks already have working. Actually Max is more like designing programs than writing them. You figure out what should happen and in what order, then drag code chunks around and connect them together.

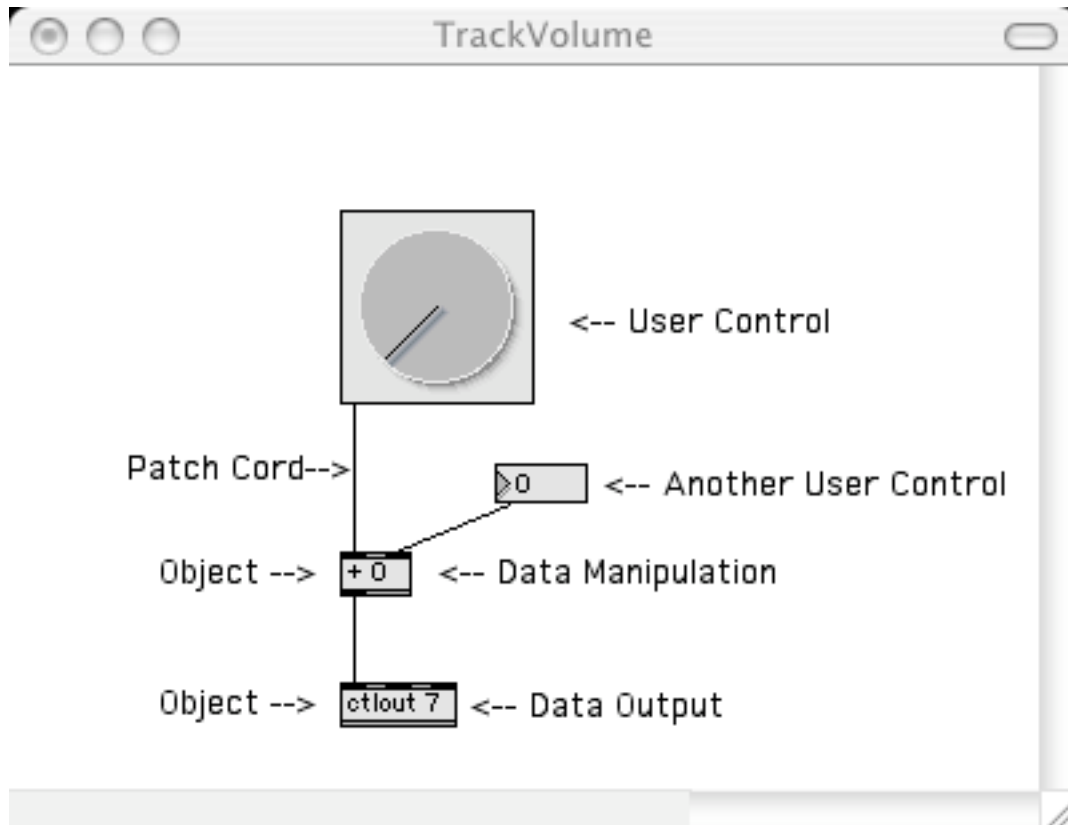
Max is easy to learn, if you are prepared to devote some time to it. There's a lot of material to read, but you can only pick up a few things from books. Most of your learning will come from experimenting and making mistakes, so be sure to spend as much time trying things as reading. Things you should read first:

- Max Fundamentals
- Max Tutorials up to 14
- My tutorials in the basics folder

You should also have the Max Reference Manual open as you work.

Patchers

MAX is designed to look familiar to composers who have worked with patchable synthesizers. What you see on the screen is a lot of boxes with lines connecting them. The boxes are called objects and the lines are patch cords. What happens is that each object represents a process. The results of the process are passed along the patch cord to the next object. Ultimately, there is an object that sends MIDI data, audio or video out into the world.



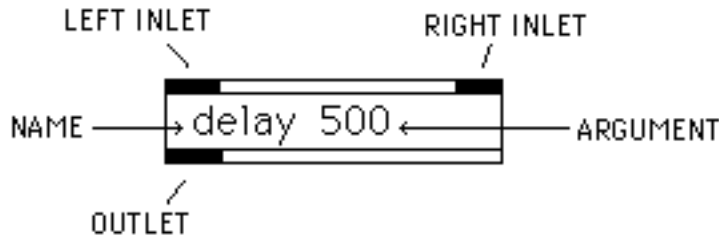
Each window full of objects is called a patcher. Several patchers may be open at once and they can all be active, even if their window is hidden.

Patchers can be saved, and then entered as an object in another patcher. There is also a patcher object, that can be opened up and filled with objects which will continue to work after the patcher object is folded up again.

The action flows from the top down. When an object is tweaked by the user or MIDI comes in, a message is sent to any connected objects, which react with messages of their own. Only one thing happens at a time, but it's all so fast it seems instantaneous. When a pathway branches, messages are sent to right destinations before left.

Objects

The basic object looks like this:



The name of the object determines what it does. There are a few hundred objects supplied on the disk, ranging in complexity from simple math to full featured sequencers. Arguments, if present, specify initial values for the object to work with. Data comes into the object via the inlets, and results are put out the outlets. Each inlet or outlet on an object has a specific meaning. This will be displayed in the lower left of the window as the mouse passes by (further details are in the manual). Usually, input to the left inlet triggers the operation of the object. For instance, the delay object (as shown) will send a bang message (see below) out the outlet 500 milliseconds after a bang is received in the left inlet. Data applied to the right inlet will change the delay time.

Messages

Data bytes sent down the patch cords are called messages, which fall into one of the following types:

int A number without a decimal point.

float A number with a decimal point.

symbol A character string¹ such as “stop” that may be understood by certain objects.

list Several of the above, separated by spaces. The first element of a list must be a number.


bang A message that triggers the action of an object.

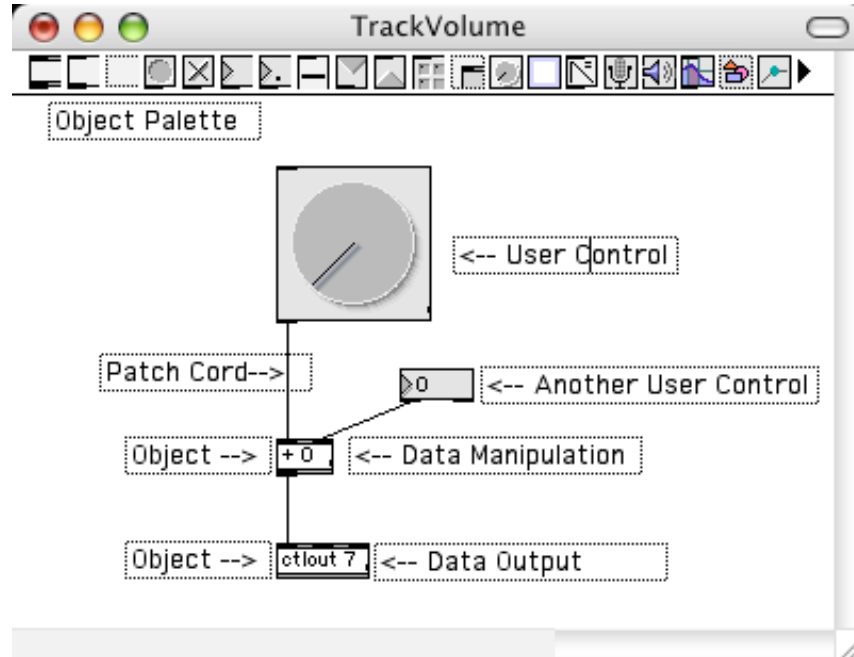
Audio signals are sent in yellow patch cords. These are little packets of data, but sent so fast as to be effectively continuous.

Jitter signals are sent via green patch cords. Jitter messages are names of matrices that hold data for jit.objects to process.

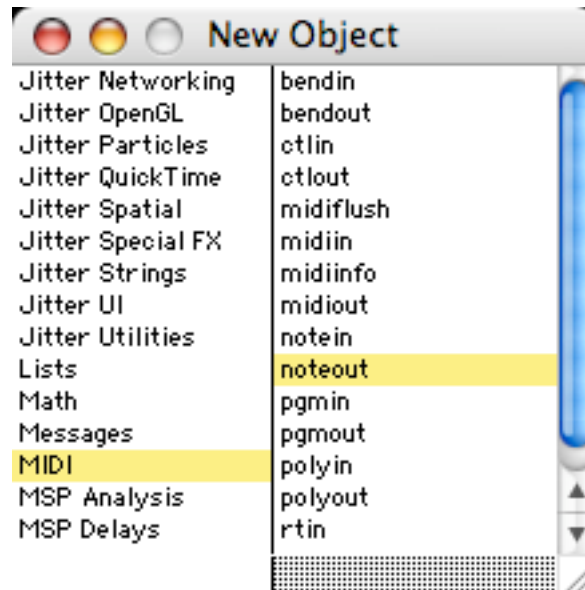
¹String is computerese for a group of letters.

Editing Patches

There is a widget  in the title bar of a patcher. If the widget is clicked (or command -e) the patcher may be edited.



The palette across the top of the window contains object icons. Click on your choice (usually the left one) and move it into position and click again.



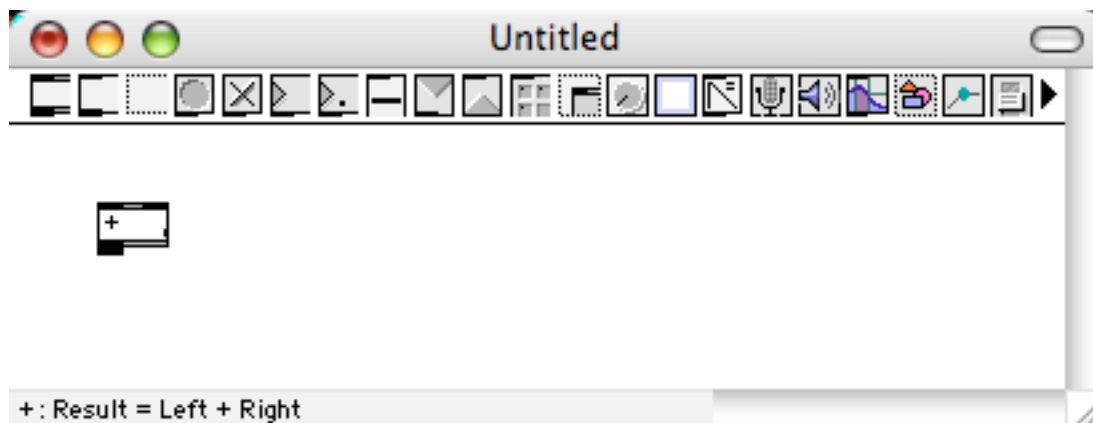
If you choose an object box, you then type the name. If "New Object List" is checked in the Options menu, you will get a dialog that allows you to select the name. This has

categories (click in the left pane, then arrow down to see more), but the most useful is "All Objects". Type the first letter of the name you are looking for and arrow to select it, then hit return.

Learning the names of objects is the most daunting aspect of learning Max. (Heck, I can't even remember the names of all the objects I have written myself.) There are over 300 stock objects and more than 100 Lobjects in the studio installations. If you are setting up your own version, you will find more than 1000 externals authored by dozens of other developers. Don't Panic! Most patches are made of the same basic objects, which I have documented in the essay "PQE's Favorites". Learn these first, then study the Max Object Reference PDF. The Thesaurus at the end is particularly useful.

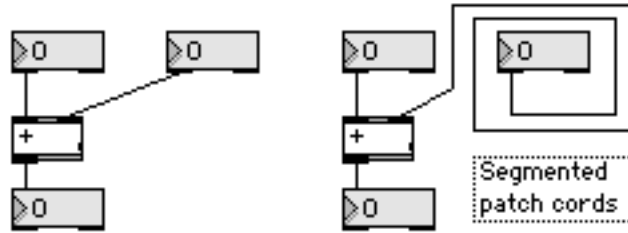
If you option-click on an object, a help screen will come up that explains how it works. This screen is a working patcher. You can unlock it and copy portions to your own window. (Just be certain to click Don't Save Changes when you close the help window!)

Enter patch cords by clicking on an outlet. You then drag a line to the inlet you want.² The function of each outlet or inlet will appear in the lower left of the window as the mouse moves over it.



There are two modes of patch cord entry. If "Segmented Patch Cords" is not checked in the options menu, you must drag a cord directly from an outlet to an inlet to make the connection. If the option is checked, clicking on a outlet will start the cord, and clicking in empty space will put a corner in the cord. Click on an inlet to finish the connection. If you find a cord stuck to the mouse with nowhere to put it, command click on empty space.

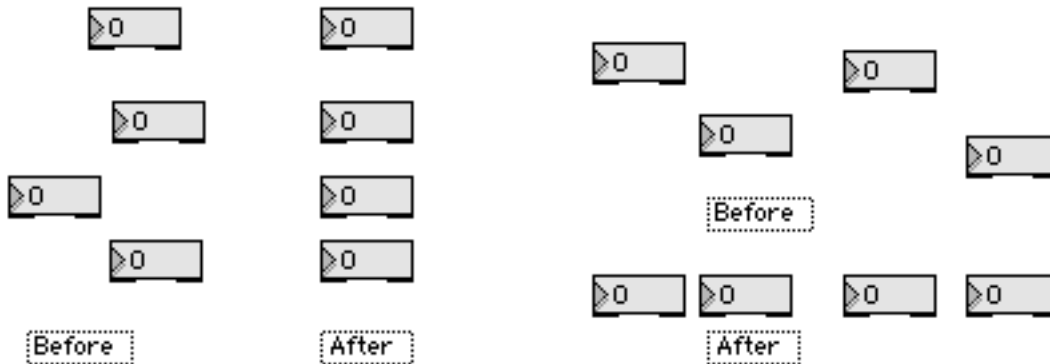
²Max won't let you make inappropriate connections.



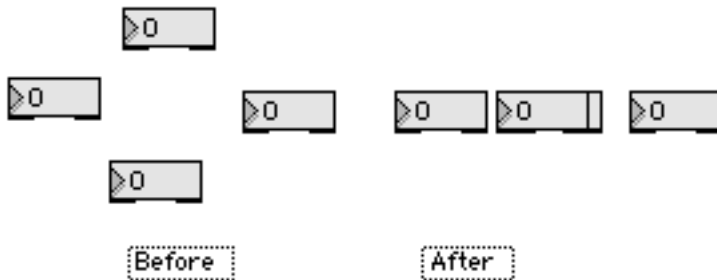
If "Segmented Patch Cords" is not checked in the options menu, you can make a segmented cord by holding the shift key as you click on the outlet.

Many connections may be made to a single inlet or outlet. If you are in segmented mode and hold the shift key when you click on an inlet, you will get a new cord from the same outlet. This makes it easy to connect from one source to many destinations.

You can select a cord by clicking on it. (Option-drag to select multiple cords). The delete key will then remove the cord. If you hit command-Y when a cord is selected, it will become segmented. If you select objects and hit command-Y, the boxes will be aligned neatly. The program chooses horizontal or vertical alignment based on the approximate line-up:



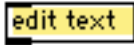
Command-Y is not perfect:



You can select objects by the usual clicking, shift clicking and drag around motions. Selected groups can be copied, cut and pasted. The patch cords will come along if both

source and destination objects are selected. Command-D will duplicate a selection, pasting it to the right and below. If you move the copy and duplicate again, the spacing set by the move will be used.

There are two selection modes for object and message boxes:



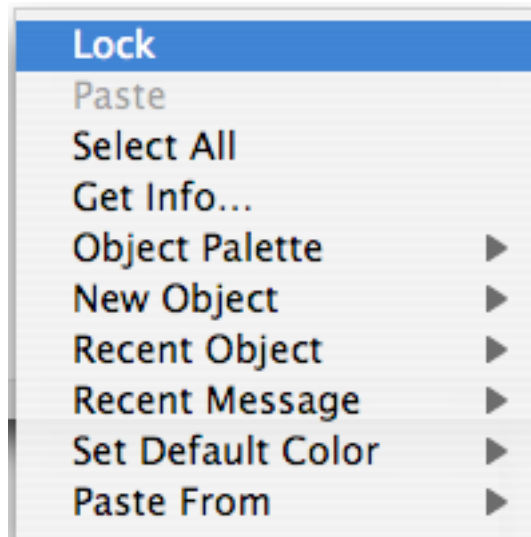
Yellow selection is for editing the contents of the box.



Gray selection is for copying the object itself.

Getting the gray selection is a bit tricky. I find it easiest to drag across a corner of the box.

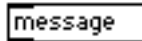
Control click, or right click on a two button mouse to get a menu of useful activities: modifications to the object clicked (a combination of the Edit and Object menus), or ways to create objects if you click over white space.



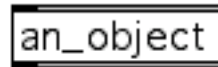
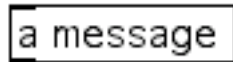
While you are editing, you cannot interact with the user controls with the mouse (The mouse moves them around). You can momentarily lock the patcher by holding the command key (ctl on PCs). Then the controls are operable.

Interface and Display Objects

Most of the objects in the edit palette are there to allow the user to control the patch. There are sliders and dials, even a cute little keyboard. The most useful are the simple ones. For instance:



The message box will contain a symbol (up to 255 characters long) or a list. Usually these are entered when the box is created. When the user clicks on the box, the message is sent out. The message will also be sent if anything is received at the inlet. The symbol “set” at the inlet will cause the message to be changed to whatever follows. Note how similar the message box is to an object box.



Using one where the other is needed is the most common mistake made by new Maxers.



Number boxes are used a lot. (There are two kinds, for ints and floats.) A number that comes in the inlet will be displayed. A bang at the inlet will send the number out. If the user clicks on the box and drags up or down, he can change the number, which is sent out in the process. The user can also click and type in the box, hitting return to complete the entry. If you select the box during editing and click on Get Info in the menu, you can customize the behavior of the box, such as setting limits on the values, or making it display pitch names instead of the number.



This thing increments or decrements number boxes. It is set up like this:



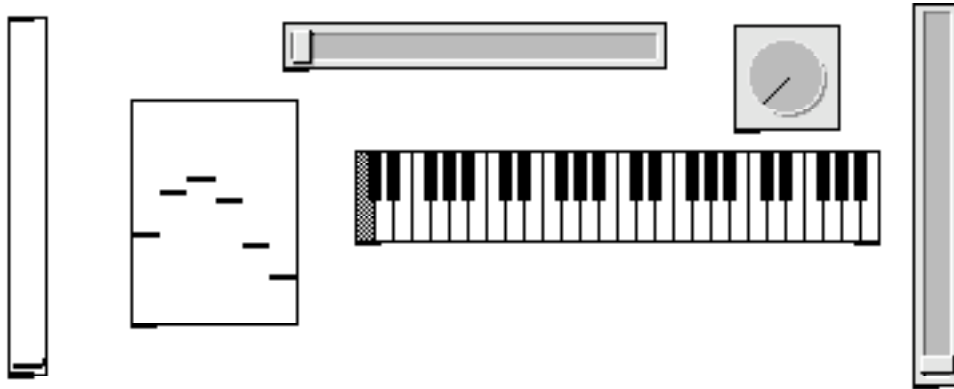
Buttons send a bang when the user clicks on them. If any message is received at the inlet a bang is sent.



When the user clicks on the toggle the X disappears and a 0 is sent. If he does it again, the X comes back and a 1 is sent. A 0 at the inlet will clear the toggle, and any other number will set it. A bang will flip the toggle to the other state. (1 or 0 is sent only when the toggle changes.)



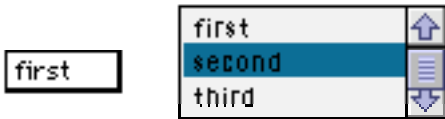
The led behaves more or less like a toggle, except it will flash and output its current state (0 or 1) if it is banged. Its state is changed by mouse clicks, explicit data, or the message "toggle". A get info window lets you change the color and flash time.



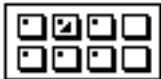
These are all variations of sliders- the user drags or clicks on the control in the obvious way, and data comes out the outlets. Get info allows you to change the range of data, and the appearance of the item. These can also give graphic representation of current data values.



The mousestate objects tells you what the mouse is up to all the time.



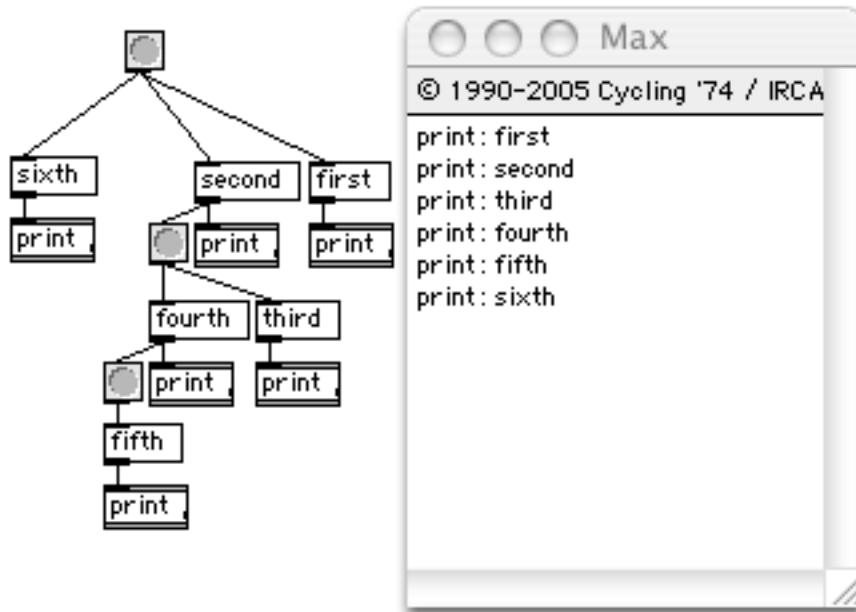
These put menus in the patcher window. There is an object called menubar that allows you to modify the menus at the top of the window.



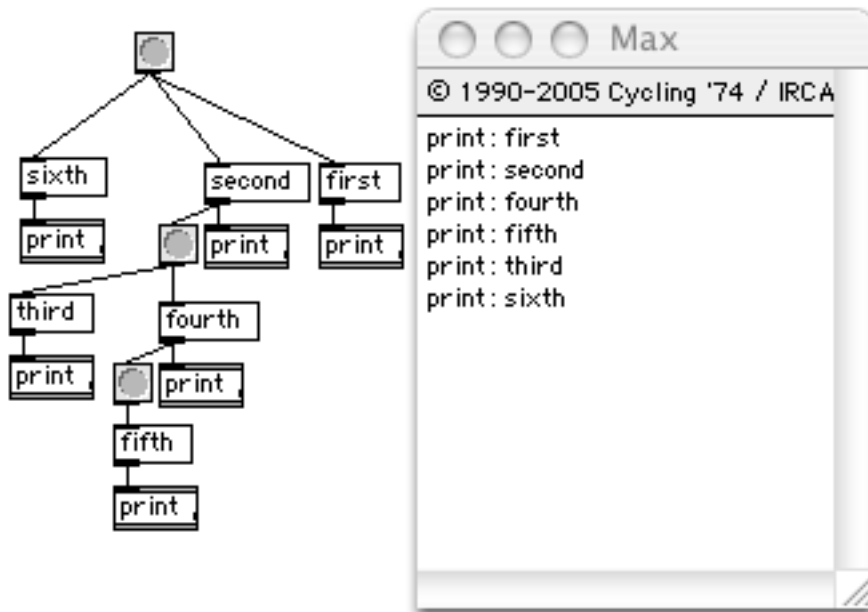
The preset object allows the user to store the state of all controls. To make this happen, you just put the preset in the patcher somewhere. If the user shift clicks on a button, all current values will be stored. Later, a click on the button will bring them back. The dot on a button indicates it contains data, the triangle indicates the most recently selected preset. A preset object with patchcords attached only affects the connected items.

Right to Left Action

When messages are sent to more than one inlet from the same outlet, the rightmost path is followed first.



Order of execution can be changed simply by moving objects around.

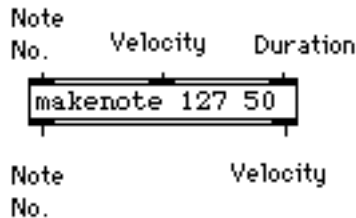


How to Get Music In and Out of Max

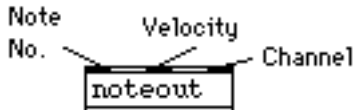
Max is a number cruncher with timing features. Max knows nothing about music. As long as you keep this in mind you should have no trouble getting interesting things to happen.

Making Notes

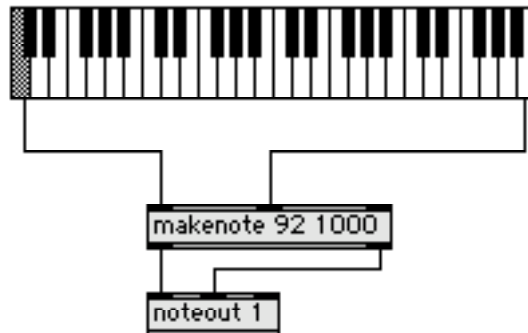
In MIDI, as you may remember, there are six numbers associated with each note: the note number, velocity and channel number for the note on, and the same three for the note off. If the note off channel number and note number do not match the ones for the note on, the note will play forever. Also remember that a note on with a velocity of 0 counts as an off.



Down toward the bottom of almost every patch, you will find a makenote object. When a note number (an int between 0 and 127) hits the left inlet, a velocity (as set by the middle inlet or the first argument) goes out the right outlet, followed by the note number out the left outlet. A few milliseconds later, as set by the right inlet or the second argument, the note number comes out again, but this time with a 0 velocity. These values are just what the next object needs:



Noteout uses the channel, velocity and note numbers provided to send a note message out the MIDI port. This patch plays notes of 1 second duration on channel 1:



Midi Input

Getting MIDI into MAX is duck soup. The midiin object gives you the raw midi data, that can be sorted out with midiparse:



Each type of channel message has its own outlet on midiparse. (Noteoff is always received as a noteon with velocity of 0.)

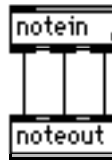
If we are only interested in one type of message, there are specialized objects. For instance, a notein object provides note numbers, velocities and channel numbers, ctlin provides control number and value (or watches just one control).



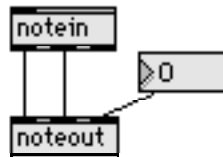
There is also xnotein if you want release velocities on note off, and xbendin for double precision pitchbends.

There are complementary -out objects for all of these.

So, this patch

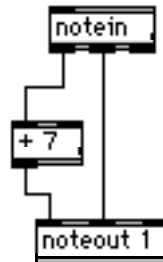


will simply pass notes through the computer. This one changes the channel:

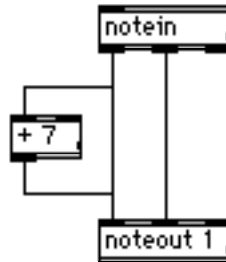


(Warning: BE SURE NO NOTES ARE PLAYING WHEN YOU CHANGE MIDI CHANNELS. Else any pending note offs will be steered to the wrong channel and the note will hang.)

We can mess with the pitch of the notes:

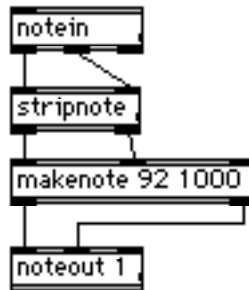


Here, transposing them up a fifth. (7 semitones.) If we make one slight change:



We'll hear the note and the fifth at the same time. We could add a number box to adjust the transposition, but we'd have to be careful. If the transposition were changed while the note was sounding, the old transposed note would never get a note off.

For many processes, we only want the note ons, without the note offs. The stripnote object does the job:

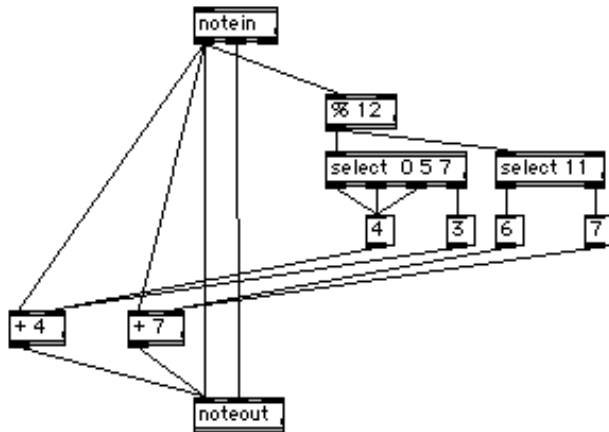


This patch holds any note you play for one second.

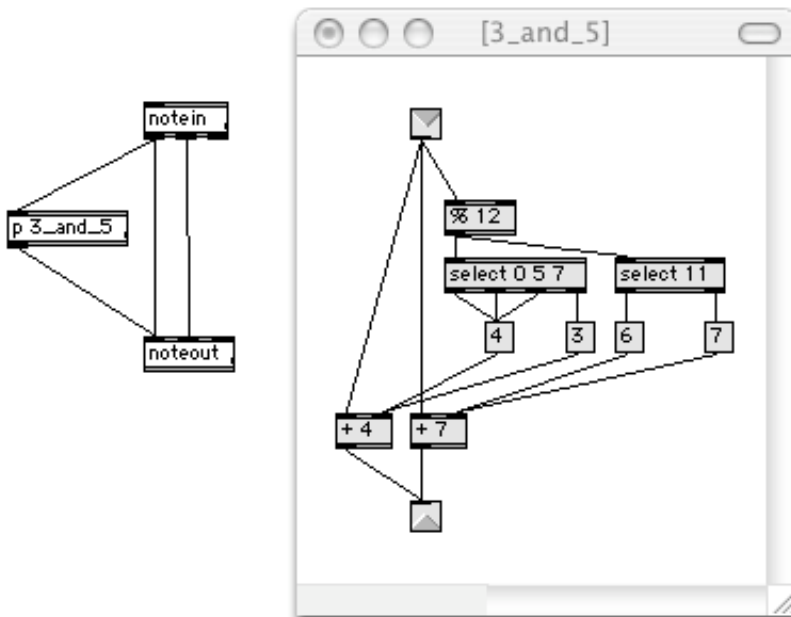
Obviously, the stuff between notein and noteout can get very complex. We'll see many examples in the following sections.

Patchers in Patchers

You will find that patchers get cluttered fast. One way to manage this is to use "encapsulation" or subpatchers. There are two kinds of subpatcher. One is made with the patcher object. Here is a messy little patch:



This creates chords appropriate to a C major scale. It is essential that the intervals added to the played note be calculated before the additions occur, so there are a lot of wire crossings to keep the selection logic on the right side of the adders. This can be cleaned up by sticking most of the objects in a patcher object. Type p and a good name in an object box. A new window opens up. You can construct part of the patch, using inlet and outlet objects to define the points when messages will be passed in and out of the patcher object.



There is a short cut to creating subpatches. Simply select the items to include (they should form a well defined group) and choose encapsulate from the edit menu. You can open the patcher object for editing by holding the command key and double clicking it. If you duplicate the patcher object, you will have two independent version of the code.

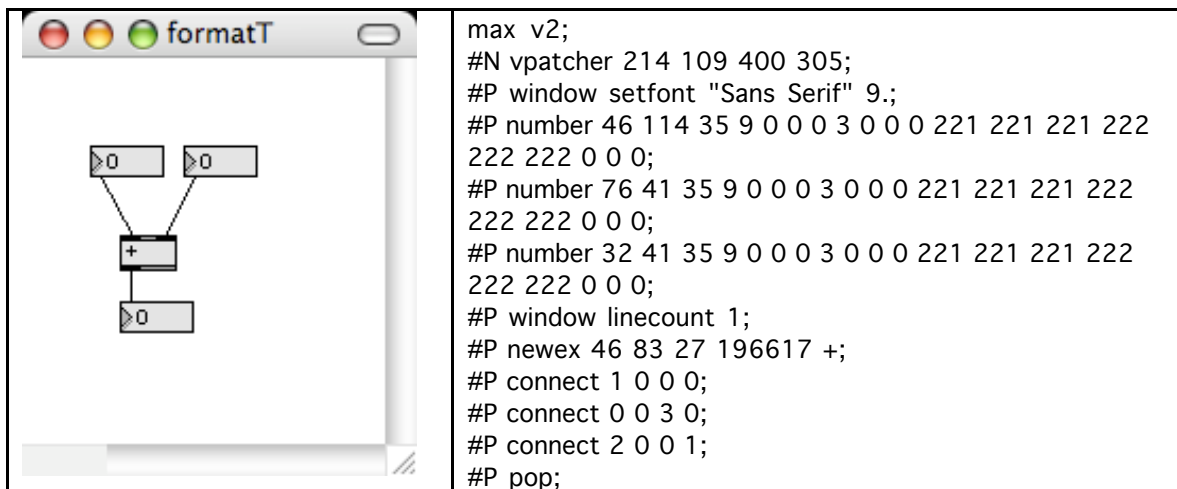
You can also save a subpatch like this to disk. Then to use it, you need only type the name in an object box. You can open a loaded subpatch by command double clicking, but you must open the original file to edit it.



The difference between the two methods is that the patcher object is one of a kind that is saved with the enclosing patch. If you duplicate it, you can edit the duplicate without affecting the original. A subpatch saved as a file can be included in any number of patchers. If you edit such a file, all instances of the subpatch will be changed.

Max File Formats

Max Patches can be saved in two formats. The usual is as a Max Binary, which has the extension .mxb or .pat in windows systems. (The .pat extension is also used by Photoshop, so there can be some confusion. Use the finder to fix these..) Patchers can also be saved as text, which makes them easy to share in email. The text format extension is .mxt or .txt.



You can convert text to a patcher by this process:

- Create and save an empty patch.
- Open it as text.
- Paste the received text in.
- Save and close the window.
- Open the usual way.