INTRODUCTION TO MAX

Revised for Max 6

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[1] 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 because we can write our own code. The problem with that 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 need. 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:

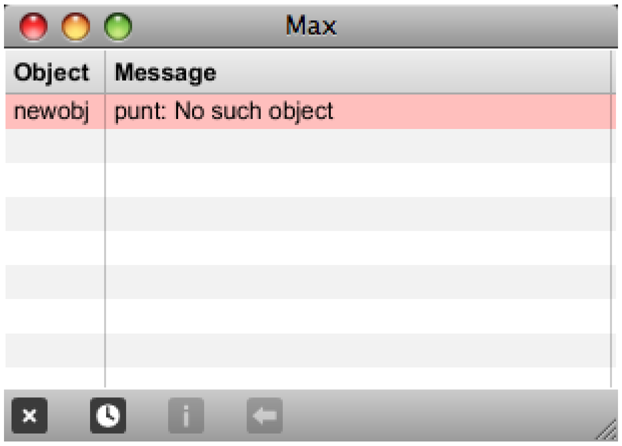
- Making Patches from the help window home screen. (Follow all of the links.)
- Max Tutorials up to Keyboard and Mouse input.
- My tutorials in the basics folder at ftp://arts.ucsc.edu/pub/ems/maxtutors/

You should also have the Max Reference open as you work. It's right there in the Patcher sidebar.

[1] The same comments apply to the art scene, just a little later with different software.

Windows

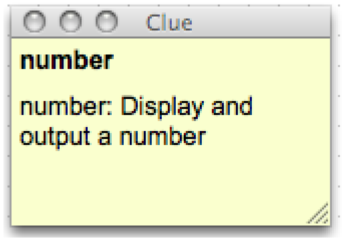
Max has several types of window. The work is done in patcher windows, but you may choose to show other windows from time to time. The Max window contains information sent from Max (like error messages) or things you might like to print.



Object	Message
newobj	punt: No such object

The X button clears the Max window.

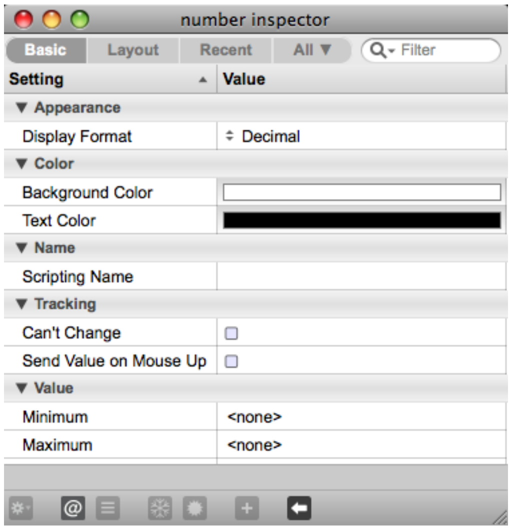
The Clue window has helpful information about anything the mouse is over. You can add your own clues.



number

number: Display and output a number

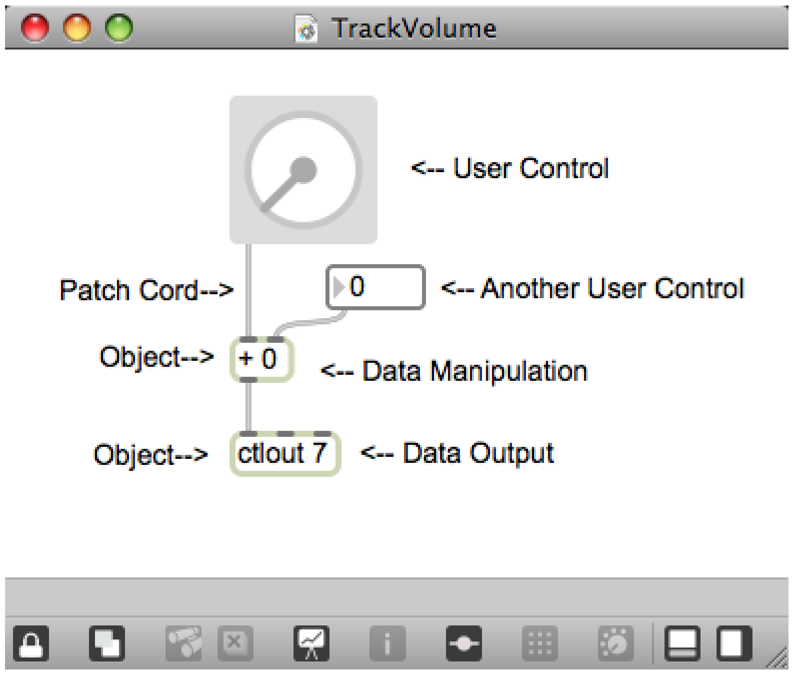
The Inspector window shows current settings of a selected object. The inspector can be a separate window or in the sidebar of a patcher window.



Setting	Value
▼ Appearance	
Display Format	Decimal
▼ Color	
Background Color	⬜
Text Color	⬛
▼ Name	
Scripting Name	
▼ Tracking	
Can't Change	☐
Send Value on Mouse Up	☐
▼ Value	
Minimum	<none>
Maximum	<none>

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.



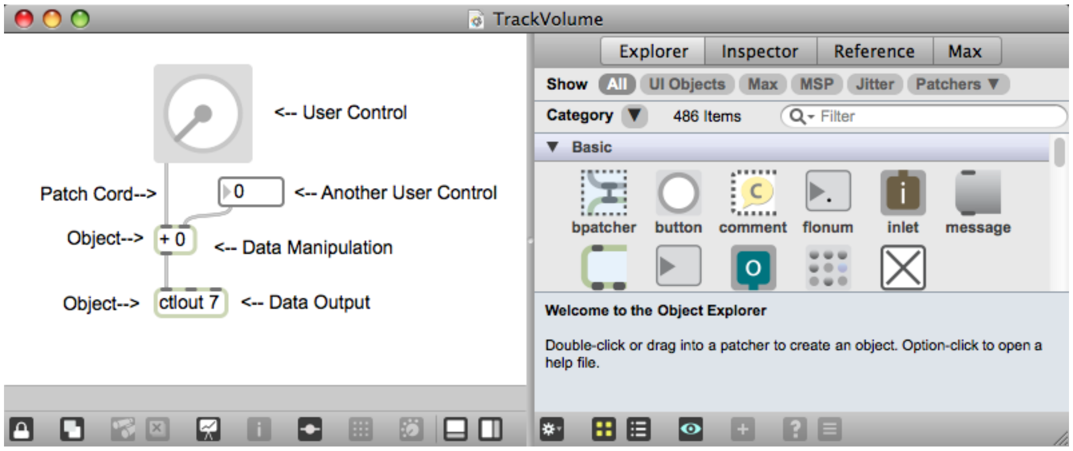
Each window full of objects is called a patcher. Several patchers may be open at once and they can all be active, even if their window is hidden.

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 a UI object is tweaked 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.)

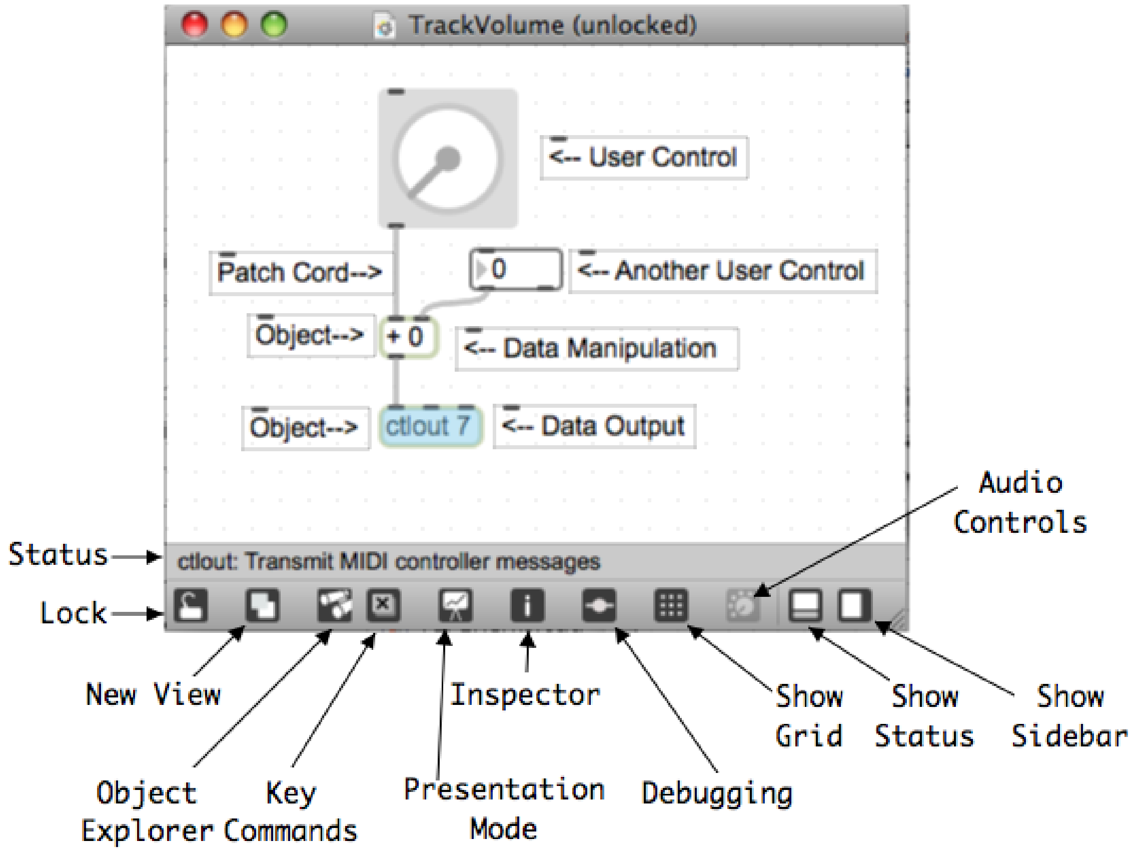
Sidebars

Patcher windows have a sidebar that can be shown by clicking this widget at the bottom left. ◻



The sidebar can show an object explorer, the inspector for a selected object, the reference page for a selected object, or the Max window.

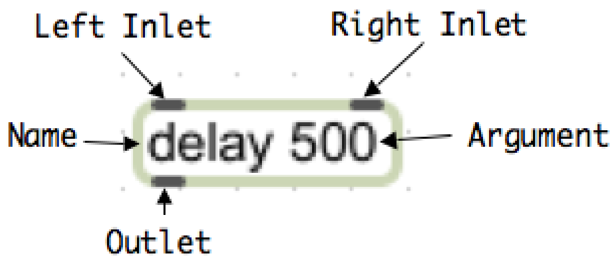
Patcher controls



There are a variety of controls across the bottom of the patcher window. These all enable functions to be discussed below.

Objects

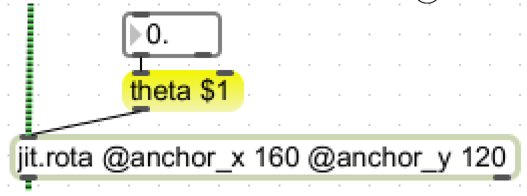
The basic object looks like this:



The name of the object determines what it does. There are a few hundred objects included with Max, 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 a flag as the mouse passes by (further details are in the documentation window). 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.

If an object has several arguments, the order of the arguments determines the meaning. For instance, a counter with arguments of 2 0 16 will count from 0 to 16 and back:

Some objects, especially those in Jitter, have a special type of argument called attribute. An attribute argument represents an internal variable and is set with two or more parts: the name of the attribute with a @ prefixed to it and one or more arguments for values of the attribute. The order in which @attributes are entered is unimportant.



In this example the jit.rota object has the anchor_x and anchor_y attributes set by arguments. You can also set attributes by sending a message that includes the name of the attribute followed by the values. In this example the "theta" attribute is being set via a message.

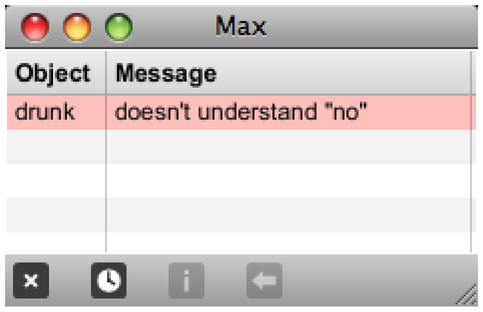
Messages

Data sent down the patch cords is 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 string[2] such as "stop" that may be understood by certain objects. A symbol may be followed by further information in the message.
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.

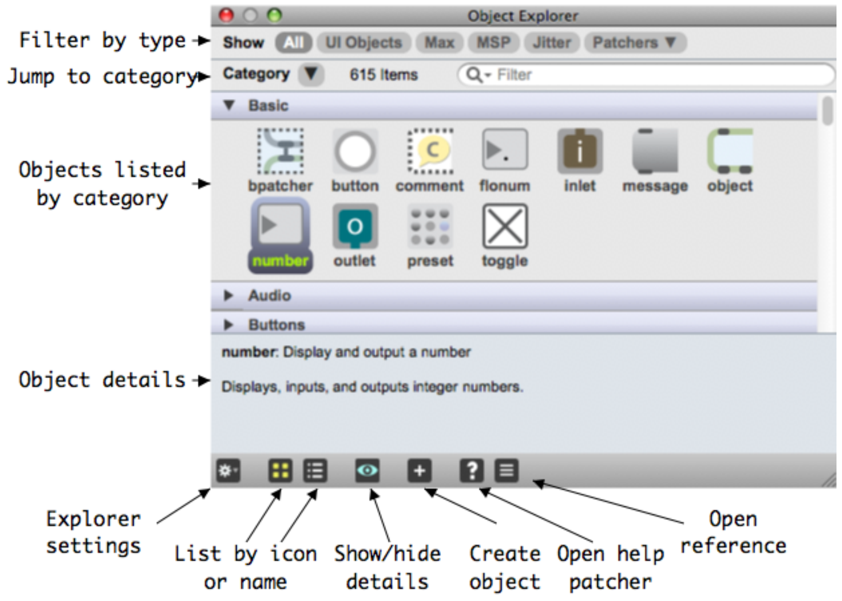
Every object responds to a variety of messages. If a message won't work, a warning will appear in the Max window.



Object	Message
drunk	doesn't understand "no"

Editing Patches

You can begin creating a patch by getting a New window from the file menu or simply cmd-n. There is a lock widget in the lower left corner of a patcher. If the widget is unlocked by a click or command–e the patcher may be edited. A click on the button or a double click in white space will open the object explorer:



number: Display and output a number

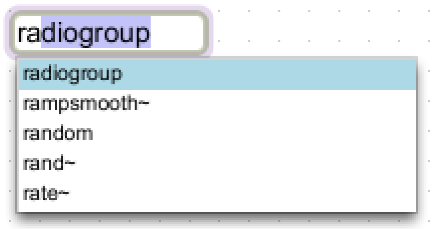
Displays, inputs, and outputs integer numbers.

[2]String is computerese for a group of letters.

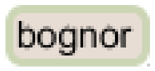
The object explorer shows all built-in objects and provides tools for finding objects if you can't quite remember the name.

- Select types to show
- Browse by category
- Scroll through the entire set
- Search by name or partial description

If you choose an empty object box, you then type the name. As you type, an auto-complete menu shows possible names based on the typing so far.



You can arrow down the menu to pick something. Hit return or tab to complete the choice, hit space to complete the choice and leave the cursor ready to enter arguments.



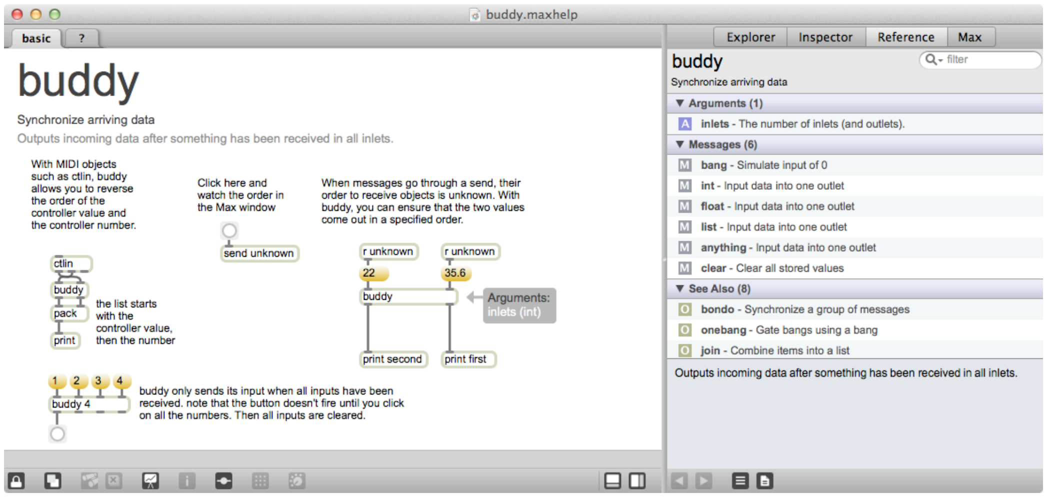
If you mistype the name of an object, you will get something like this and an error message in the Max window. The lack of inlets and outlets is the clue.

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!)

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 600 stock objects and more than 100 Lobjects in the studio installations. If you are setting up your own version, you will find thousands of 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 Objects by Function categories in the help window. If you have a need and want an object to fill it, use the object thesaurus. (It will take a while to open, and may be blank until you nudge the scroll bar.)

Help Files

Once you have an object, you can option click on it to open the help file.



buddy.maxhelp

buddy

Synchronize arriving data

Outputs incoming data after something has been received in all inlets.

With MIDI objects such as ctlin, buddy allows you to reverse the order of the controller value and the controller number.

the list starts with the controller value, then the number

Click here and watch the order in the Max window

When messages go through a send, their order to receive objects is unknown. With buddy, you can ensure that the two values come out in a specified order.

Arguments: inlets (int)

buddy only sends its input when all inputs have been received. note that the button doesn't fire until you click on all the numbers. Then all inputs are cleared.

buddy

Synchronize arriving data

▼ Arguments (1)

inlets - The number of inlets (and outlets).

▼ Messages (6)

bang - Simulate input of 0

int - Input data into one outlet

float - Input data into one outlet

list - Input data into one outlet

anything - Input data into one outlet

clear - Clear all stored values

▼ See Also (8)

bondo - Synchronize a group of messages

onebang - Gate bangs using a bang

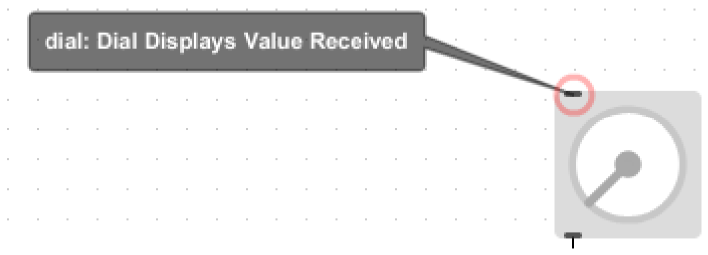
join - Combine items into a list

Outputs incoming data after something has been received in all inlets.

The help file is a working patch that illustrates the use and features of the object. You can interact with it, and unlock it and copy parts into your own patch to experiment with. You can even modify it and save the changes! (On your own copy, please.) There is a brief reference in the sidebar, but if you click on the question mark tab you can go to a very detailed reference page. (You can also get to this stuff by a right click on the object.)

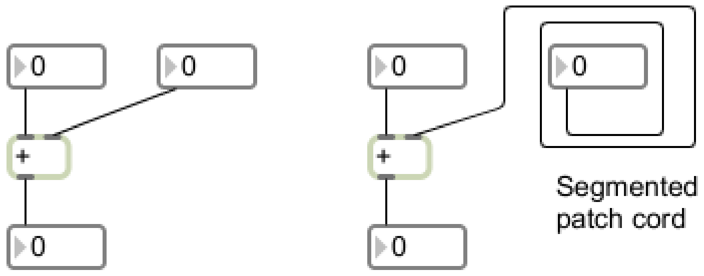
Patching

Enter patch cords by clicking on an outlet or inlet. You then drag a line to the inlet or outlet you want.[3] The function of each outlet or inlet will appear in a flag as you mouse over it. A circle helps identify the inlet that is about to be connected.



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 (or inlet to outlet) to make the connection. If the option is checked, clicking on an in or outlet will start the cord, and clicking in empty space will put a corner in the cord. Click on a target to finish the connection. If you find a cord stuck to the mouse with nowhere to put it, command click on empty space.

[3]Max 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. You can reshape a cord by selecting the cord and dragging segments.

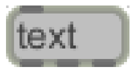
Many connections may be made to a single inlet or outlet. If you are in segmented mode and hold the shift key when you click on a target, you will get a new cord from the same outlet. This makes it easy to connect from one source to many destinations and vice versa.

You can select a cord by clicking on it. (Option-drag to select multiple cords). The delete key will then remove the cord. A selected cord has green and red "handles" on the ends– these can be used to reconnect the cord. If you hit command-Y when a cord is selected, it will become segmented[4]. 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. If you click on the precise center of the cord (a little diamond guides you) you get a menu to let you perform various operations, including temporarily disabling the cord.

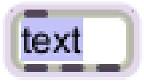
Modifying objects

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:



Complete selection is for copying the object itself. (Colors vary according to your preferences.)

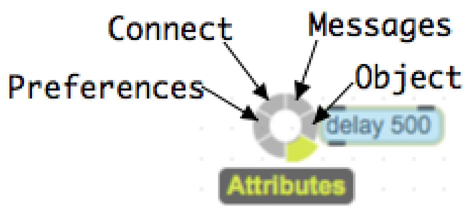


Partial selection is for editing the contents of the box.

The first click on an object selects the object, and a click (or enter) on a selected object selects the contents for editing. Further clicks place a cursor, select a word or the entire contents.

[4] Shift-Cmd-Y will attempt to route the cord around intervening boxes.

Information about an object can be accessed via the object wheel. If you hold the mouse over the left end of the box, a circular set of buttons appears[5]:

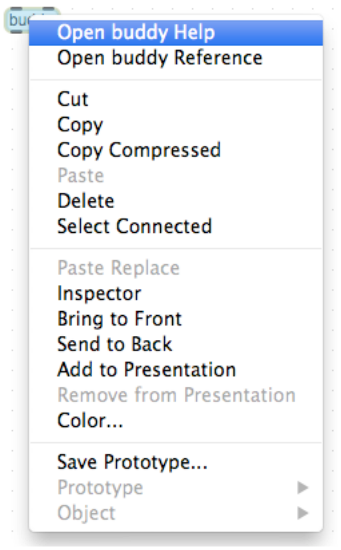


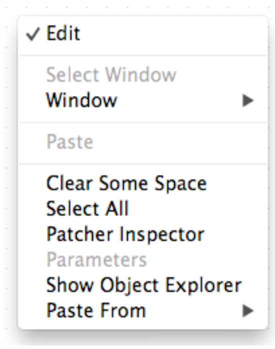
By clicking a button, you can see:

- The current state of all attributes
- The object menu to open inspector, help file, reference, or to lock the object position.
- A list of messages the object responds to-- if you click a message, you will get a message box with that message attached to the inlet.
- The function of each inlet (connect).
- Preferences for the wheel itself.

The button the mouse is over lights up with the name displayed.

Control click or right click 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 controls with the mouse (The mouse moves them around). You can momentarily lock the patcher by holding the command key (control on PCs). Then the controls are operable.

[5] This thing can easily get in the way. You can set a preference to only show a tiny version, which will bring up the full wheel when clicked.

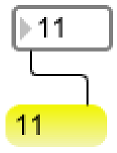
Designing Patches

Most patches include a way to input data, some manipulations of the data, and a way to output the results.

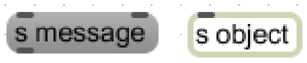
Interface Objects

The UI objects filter in the explorer will reveal everything needed to control the patch.
There are sliders and dials, even a cute little keyboard.
The most useful are the simple ones. For instance:

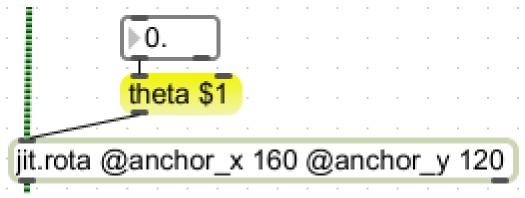
The message box will contain a symbol (up to 255 characters long) or a list. Usually these are entered when the box is created. When the user clicks on the box, the message is sent out. The message will also be sent if anything is received at the left inlet. A message in the right inlet box will be displayed in the message object, but not sent along.



Note how similar the message box is to an object box (especially with the default grey color).

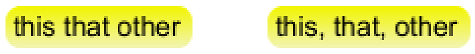


Using one where the other is needed is the most common mistake made by new Maxers. I generally set the message box color to bright yellow to improve the distinction.



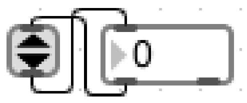
Message boxes may contain tokens of the form $1, $2 etc (to $9). When a value is sent to a message box, it replaces the $1 token and the message is sent along. If a list is sent to a message box the first item in the list replaces $1, the second replaces $2 and so on.

A message box may contain a message of many parts, or multiple messages separated by commas:



Number boxes are used a lot. (There are two kinds: ints and floats. Floats have decimal points.[6]) A number that comes in the inlet will be displayed and passed along. (The message "set 2" will change the display to 2 without causing any output.) 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 Inspector[7] in the object 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. The inspector is easier to manage if you use specialized tabs instead of all.

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



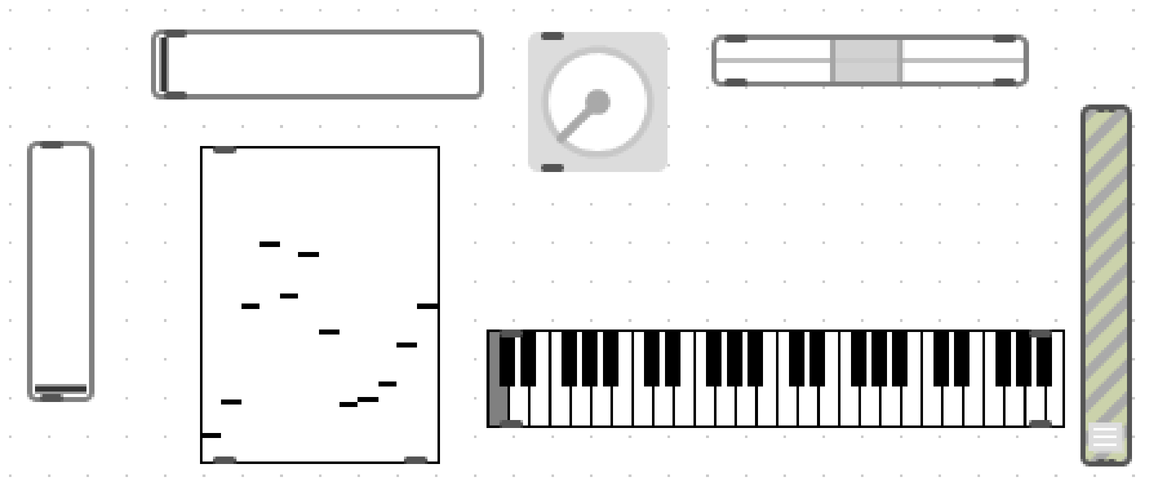
Buttons send a bang when the user clicks on them. If any message is received at the inlet a bang is sent.

When the user clicks on a 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 as it passes through. A bang will flip the toggle to the other state.

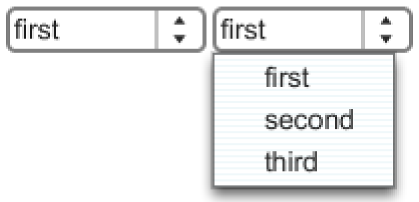
The led behaves more or less like a toggle, except it will flash and output its current state (0 or 1) if it is banged. Its state is changed by mouse clicks, explicit data, or the message "toggle". The inspector window lets you change the color and flash time.

[6] But typing a decimal point into an integer box won't change it to float.

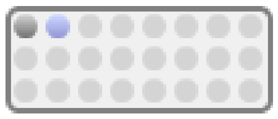
[7] Or hitting cmd-I, the inspector button at the bottom of the patcher or the little blue circle that turns up if the cursor is over the left end of the object.



These are all variations of sliders- the user drags or clicks on the control in the obvious way, and data comes out the outlets. The inspector allows you to change the range of data and the appearance of the item. These can also give graphic representation of current data values. They can be reshaped by dragging the lower right corner.



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.

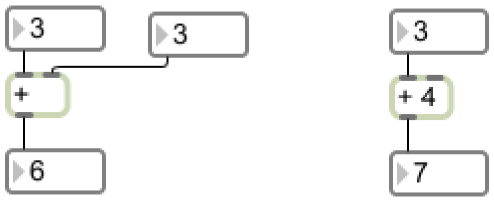


The preset object allows the user to store the state of all controls. To make this happen, you just put the preset in the patcher somewhere. The grey circles are buttons. If the user shift clicks on a button, the current values of all UI objects will be stored[8]. Later, a click on the button will bring them back. The darkened grey buttons contain data, the blue button is the most recently selected preset.

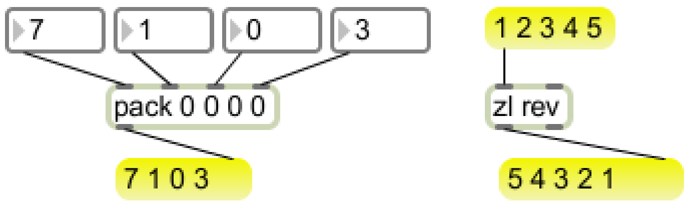
Objects to do things

Data manipulation probably involves doing math. There are many math objects, and they all work in pretty much the same way. When a value is received in the left inlet, it is modified using the value most recently received in the right and the result is sent out the left outlet. If nothing has been received in the right, the object argument is used.

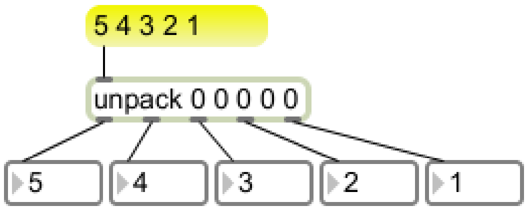
[8] There are ways to make exceptions. Check the preset object help file.



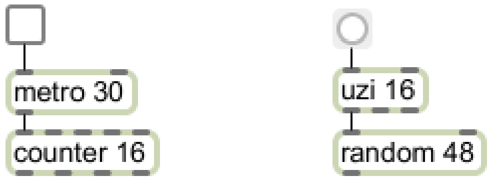
When we work with lists of data, we are concerned with assembling lists and processing them:



And later taking them apart:

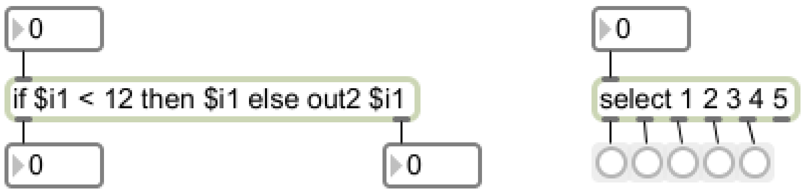


Objects are available to generate data, work with files, react to inputs and pretty much anything found in any other language. One thing you won't find is "for" and "while". For repeating actions we use metro or uzi:



Metro produces a continual stream of bangs at the interval specified in the argument. Uzi produces a defined number of bangs as fast as possible. Metro is used for steady operations like playing beats, uzi is used to generate the impression of simultaneous action.

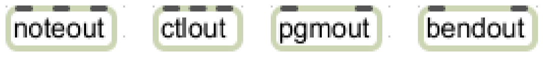
To make decisions, we use if or select:



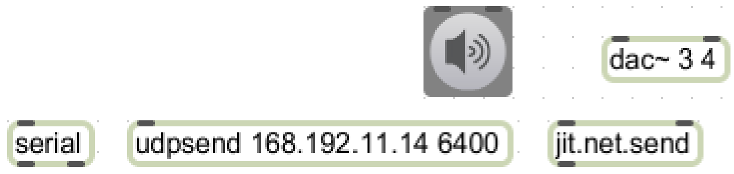
If makes either/or choices, select responds to multi-valued inputs.

Data Output

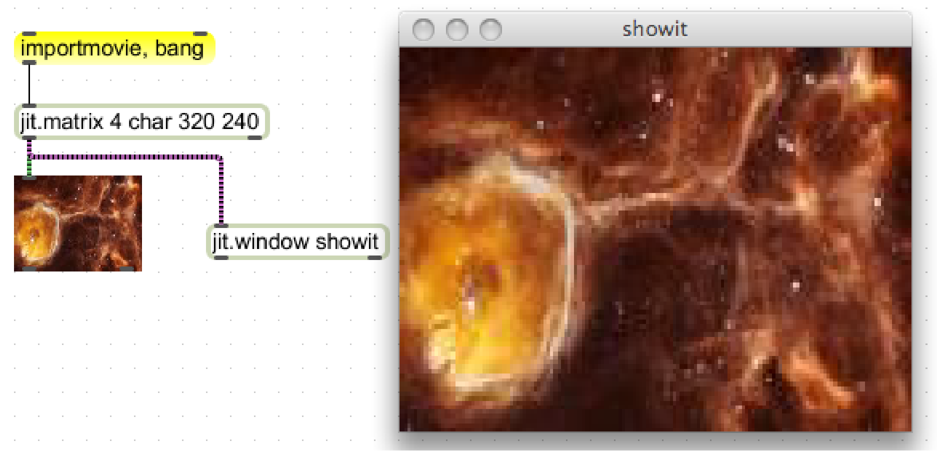
Max can produce output in a surprising variety of forms. Of course all varieties of MIDI messages are supported:



There's audio output and various communications protocols:

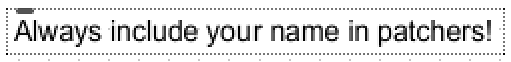


And Jitter displays:

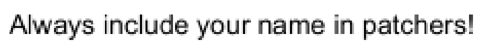


An object to do nothing

We often need to include labels, instructions, and reminders of what some parts of a patch do. This is done with comments, which just contain text. When editing they look like this:

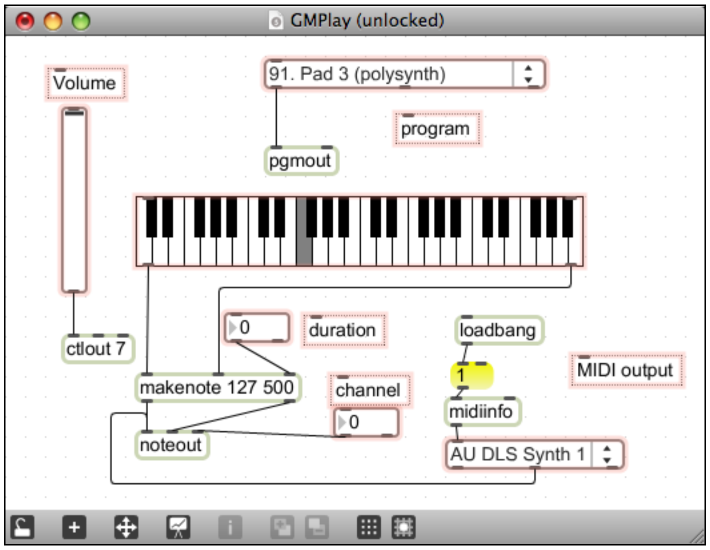

Always include your name in patchers!

When the patch is locked, they look like this:

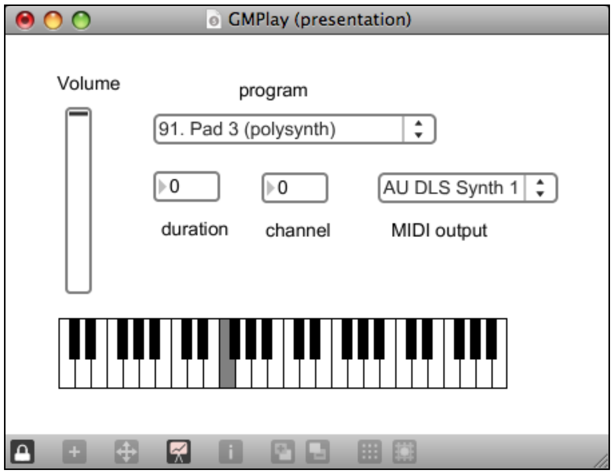

Always include your name in patchers!

You should always put enough comments in a patch to make it possible for a stranger to understand it. Chances are, that stranger is you, in about 6 months.

Presentation Mode



Even a simple patch can be awkward to use. More specifically, patches can be laid out for ease in design (top to bottom flow, short cords, associated items grouped together) or ease of use (intuitive layout, controls well labeled, most used in central locations). With Max we can have both. In patching mode, the interface items can be marked to show in the presentation (note the pink outlines). In Presentation mode, only the marked items show, and they can be rearranged for convenient use.



Power Patching

Once you get to know the repertory of objects, you can take advantage of some hot keys to place items in the patch window. For instance, hitting the n key will drop a new object box at the mouse location. (As long as there is no editing in progress. I usually click in white space to make sure.) Other keys produce other objects:

- n new object, ready for name entry
- b button

- c comment
- f float number box
- i number box
- j object box with jit. pretyped (for jitter objects)
- l object box with live. pretyped (for "max for live" objects)
- m message box
- p the object explorer
- P (shift-p) a new object selected for presentation.
- t toggle
- x a window explaining all of these

There are also command key combos for almost any operation on objects, a layout grid for those who like tidy patchers, and scheme for adding watchpoints to cords for debugging.

Performance Issues

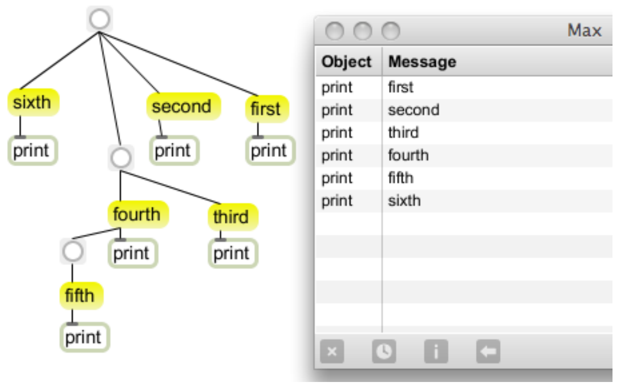
Max usually runs as fast as the computer can. The underlying code is written in precompiled C, which is linked as the patch is built. The code is running as soon as you connect the cords. Some Jitter operations may bog down, but those are usually very involved processes that would be slow in any language. Max is usually 10 to 20 times faster than the equivalent Java.

Things to watch out for

Like any programming language, Max has its tricky aspects:

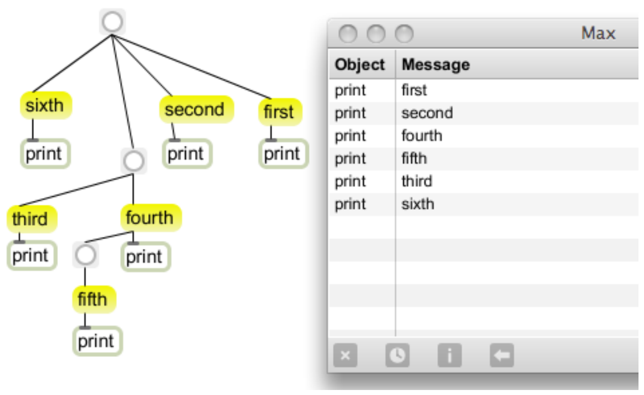
Right to Left Action

The subtlest issue in Max programming is the order of message flow. When messages are sent to more than one inlet from the same outlet, the rightmost path is followed first. Sometimes this results in calculations being performed with old data or other problems. The rules are right outlets happen before the left of the same object, and if two inlets are connected to the same source, the right one (and everything attached to it) is triggered first.



```
Object  Message
print   first
print   second
print   third
print   fourth
print   fifth
print   sixth
```

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

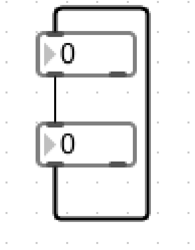


Object	Message
print	first
print	second
print	fourth
print	fifth
print	third
print	sixth

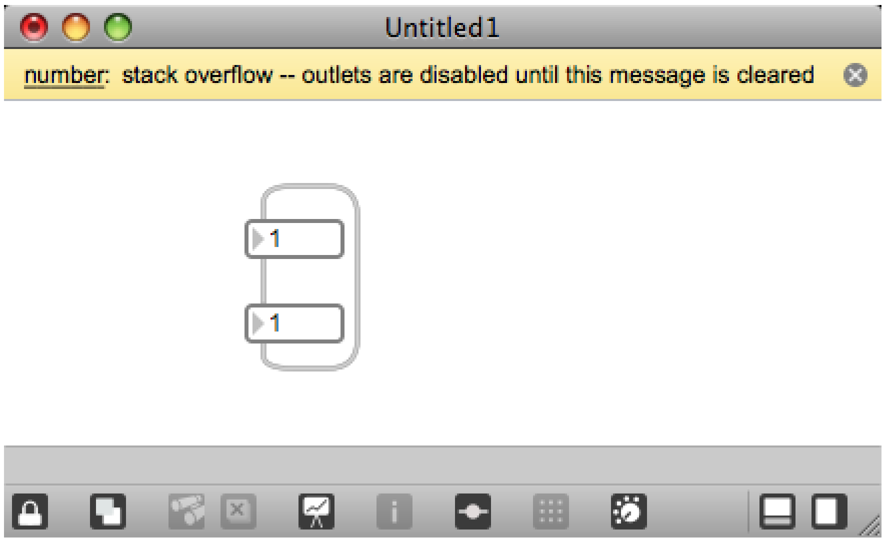
Look at the message "third".

Stack Overflow

This patch will bring Max to its knees:



When ether of the number boxes is changed, it will send its data to the other one, which will send it right back, resulting in the following:



number: stack overflow -- outlets are disabled until this message is cleared

This halts Max (the alternative would be to lock it up, requiring a force quit.) You need to find and fix the problem before restarting. You aren't likely to do something this blatantly wrong, but feedback can be produced by subtle means.

Circular loading

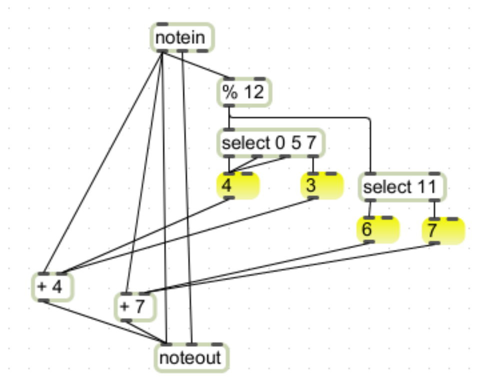
You can embed a patcher in another patch by putting its file name in an object box. If you put a patcher's own name in an object box, you create a circular reference. When you load the patch, it will load the patch contained in the object, which will load the patch contained in the object...... You will get an error message when you open the file and the self reference will be treated as a bogus object. You probably won't do anything this blatant, but you get the same effect if you ask for a patch that asks for the enclosing patch.

Missing objects

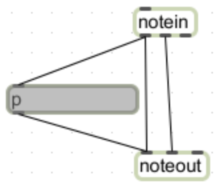
Things like audio files opened in a patch need to be where Max can find them. There is a file preferences dialog under the Options menu to tell Max where to look. These folders are searched every time Max opens, so don't just point to your hard drive. If a name is used twice in the file paths (including duplicates of Max objects) this search will report problems.

Patch Clutter

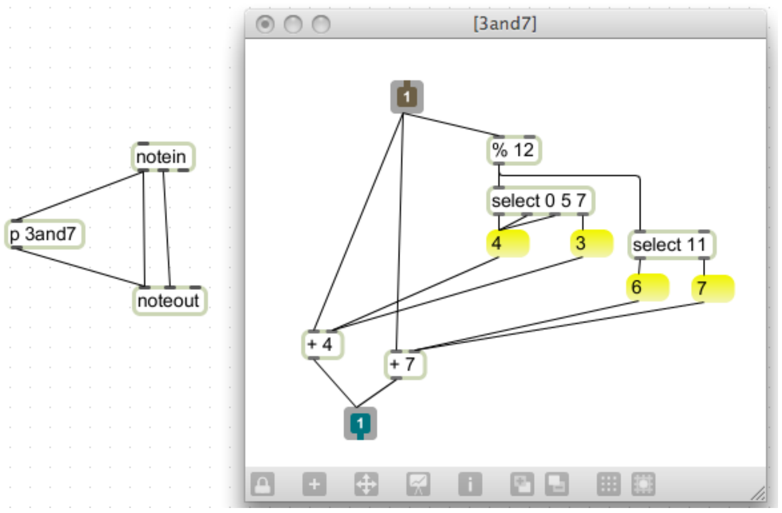
After a bit of work, Max patches often become very confusing, with dozens of objects and cords stretched all over the place. The answer to the problem is encapsulation, which allows parts of patches to be hidden away yet continue to work. There are two kinds of subpatcher. One is made with the patcher 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. Select everything but the notein and noteout objects and choose encapsulate from the edit menu.

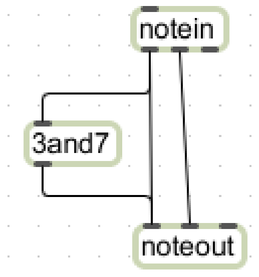


Here is the result. "p" is a short form of patcher, the object used to contain encapsulated code. Add a descriptive name to the p object. You can see the enclosed code by double clicking with the command key down.



Note the small objects with a 1 in them. These are an inlet and outlet- the number corresponds to the order as seen from the outside. If you duplicate the patcher object, you will have two independent versions of the code.

You can also save a subpatch like this to disk. (The documentation calls a saved subpatch an abstraction.) Then to use it, you need only type the name in an object box. You can open and look at a loaded subpatch by command double clicking, but you must open the original file to edit it. (Option click in the title bar.)



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 and save it, all instances of the subpatch will be changed.

Max File Formats

Max changed file formats in version 5. Version 5 can open files made in the previous format, but cannot save them that way. Versions earlier than 5 cannot read the new format. The old versions had a file extension of .mbx, .pat or .txt. They often required conversion in the process of moving from Mac to PC. The new version uses an extension of maxpat, which is text internally[9]. This is compatible with both Max and PC. (As long as all objects used are compatible. Some third party externals are limited to one platform

[9] Other extensions include ,maxhelp .jxf or .jxs. Externals are .mxo on the Macintosh.

or the other.) You can open maxpat files as text and copy the contents to email. To convert this back to a Max patch:

- Create a new empty patcher and save it.
- Open the new patcher as text.
- Paste the text from the original over the text in the new patcher. (Be sure you get all of the curly braces.)
- Save it, then open in the usual way.

You can manipulate this text directly if you care to study exactly what all of the statements mean. They are documented in the reference for the thispatcher object.

There is also a compression feature for emailing patches. Just select the part of the patch you want to send, then pick Copy Compressed from the edit menu. Something like this will wind up in the clipboard:

```
<pre><code>
----------begin_max5_patcher----------
186.3ocSO1rCBCBDD9L7TP1yUCkDiIdymCSSCTQECEZnzjpM8cW9qpWVBeyr
CCKXDHryxQfbhbgfPKXDJgh.T4NB54ycZ9XxFXl5ERGTkktYMdCuWljN6Tb8
lRvmcxqk9zZrBMi7uFj42DTFOTQ.A2bGHM+k5n5cxSMaOsfG39tGJy8Vmrym
2mQixASGiGGnwIKL+F0.2EpmW5ZkFtPmhj9qhJyVCqKP00zewJdtiAQzJFGG
jF7J9CftIiD1
-----------end_max5_patcher-----------
</code></pre>
```

Paste this into a message and email it.

If you get one, copy everything between <code> and </code> to the clipboard, then choose New File From Clipboard in the Max file menu.

Sharing

You don't need Max to run a finished patch.

Max Runtime.app

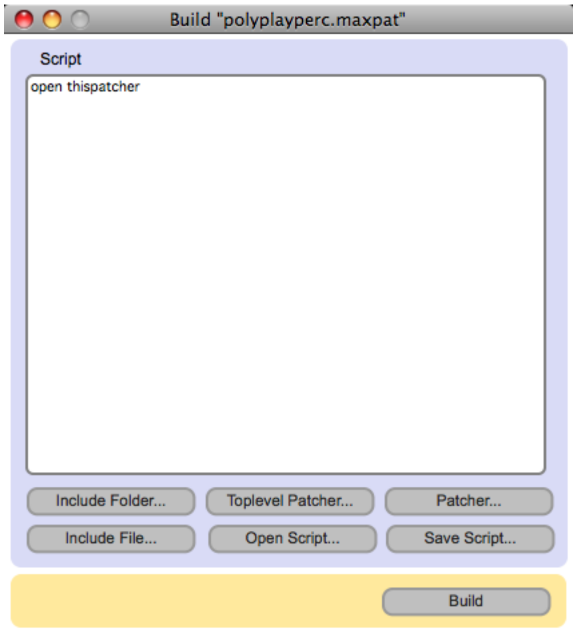
When you installed max, you also got something called Max Runtime. This is a free version of max that does everything but edit patches. The easiest way to install this on another computer is to download Max then delete Max.app from the Max 6 folder in applications. (Unless you want a 30 day trial). Don't uninstall, because runtime needs the same support files that Max proper does. If your patch requires third party externals, be sure to copy them to the new Cycling74 folder. Now any patch will run. The only limit is runtime will not save patches, so some operations like setting up presets will not be possible without extra effort.

Collectives

If your patch requires several supporting files-- externals, graphics, sounds and so forth, it can be a pain to set up using runtime. Complicated patchers can be managed by bundling them as collectives. Choose Build Collective/Application from the file menu. You get these options

- Include folder-- easy if everything is already together
- Toplevel patcher-- add a patcher to show the user. (Be sure to set Open in Presentation in the patcher inspector if that is appropriate.)
- Add patcher-- if more patchers are desired.
- Include file-- for media and so forth

As you do this, you will see a script built up in the window. You can save this script and open it later if you want different versions of a collective or the same collective with updated files. That's a good idea, because we usually need to make final tweaks to a collective two or three times before everything is right.



```
open thispatcher
```

Finally, the build button lets you name and save the collective. A collective should run under either version of runtime, Mac or PC.

Projects

For really ambitious endeavors, you can combine files into projects. What this gets you is a browser to provide quick access to everything from a single point. You could gather together all of the pieces you need for a concert, for instance. Projects can be easily converted to collectives or apps.

Standalone Applications

You can also bundle Max Runtime with your stuff to make a completely self contained application. Just choose that option as the file format when you click the build button. This will be an app that matches the system you are using- Mac or PC. If you need both, you'll have to run the process on both systems.