What is Max?

> "Max is a graphical music programming environment for people who have hit the limits of the usual sequencer and voicing programs for MIDI equipment."
>
> —Miller Puckette, Max reference manual, 1988

Max was conceived in 1986 as a project for producing interactive music at IRCAM (Institut de Recherche et de Coördination Acoustique/Musique) in Paris. The original author was Miller Puckette. Max became a commercial product from Opcode Systems in 1991 with further development by Puckette and David Zicarelli. Cycling '74 became the publisher of Max in 2000. Since that time, Max expanded to include audio data (with the introduction of MSP) and image/matrix data (with the introduction of Jitter).

Max lets you control your equipment in any way you want. You can create applications for composing, improvising, and ordering or modifying media—anything you can imagine doing with a computer. Because Max turns all control information into a simple stream of numbers, you can "patch" anything to anything else.

Max provides you with a high level, graphical programming language. Programs are "written" using graphical objects rather than text. This reduces the need to learn a lot of arcane commands and syntax, and it provides a clear and intuitive way to write programs simply by connecting objects to each other.

Max takes care of all the low level programming tasks for you. It will trigger events at any arbitrary time in the future, interface to MIDI and other communication protocols, and perform useful logical operations.

Applications made with Max run in real time. Because of its speed, Max enables you to write programs that generate music instantly based on what you play, or that modify your performance as you play.

Max is based on the C programming language. Max provides a simple yet versatile, high level, graphical language which is itself written in C, but will be easy to use for those familiar with almost any other programming language, or even for those who have never programmed before. For those who are fluent in the C language, however, Max can be combined with C code that you write. So, if there's something you need to do that Max can't do—and Max can do a lot—you can write your own Max objects in C. The release of Max 4.5 further extends Max's capabilities by supporting Java and Javascript.

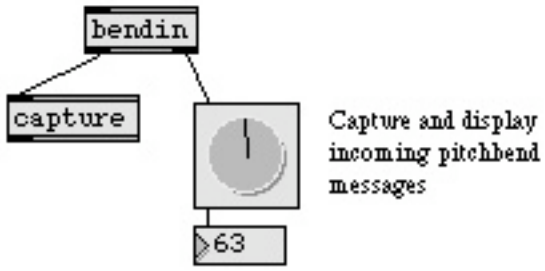
Introduction

This chapter briefly describes the Max application and introduces basic terms that will be used throughout the manuals.

If you have relatively little experience with computer programming or MIDI, you might not immediately understand everything you read in this chapter. Never fear. The chapters in the **Max Tutorial** will cover topics in much greater detail, starting from square one. Max is a complex application and we don't expect everyone to "get" everything immediately. This overview will simply familiarize you with the application and its basic elements.

Programming With Objects

When you work with Max, you'll be developing applications *graphically* using objects, which will appear on your screen as boxes that contain either text or icons. You connect objects together to create a working program.

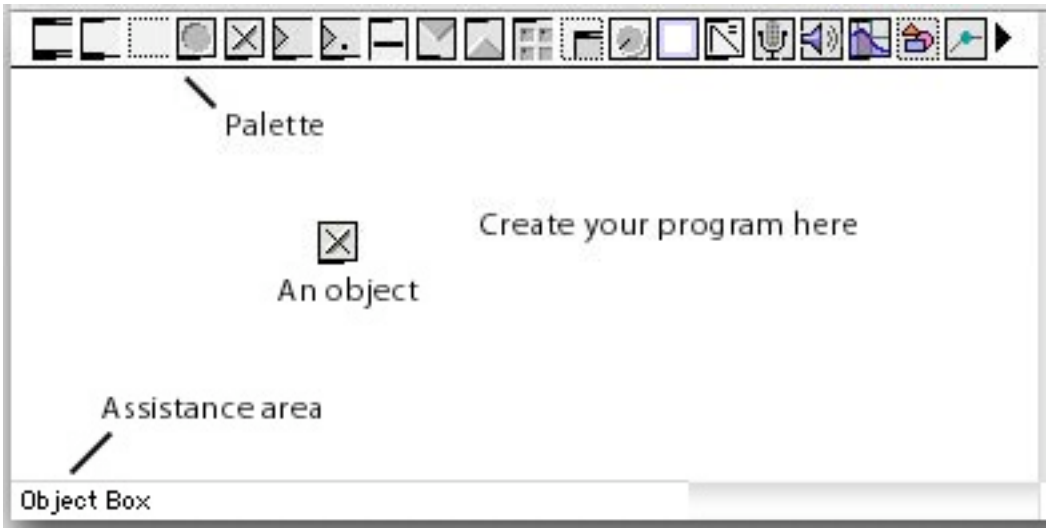


A simple program written graphically in Max

Max has about 200 different objects (and MSP adds about 200 more for audio processing, and Jitter adds an addition 140 objects for matrix and video/image processing), each of which performs one or more specific tasks. In addition, you can write your own program in Max, save it, and then use that program as an object inside other programs that you write.

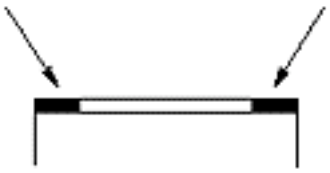
Overview

You create a Max program in a *Patcher* window, which works much like a drawing program. You select an object from a palette, then click where you want the object to go.



Objects are different types of boxes, like this:

Objects have *inlets* at the top, used to receive information from other objects...



…and *outlets* at the bottom, used to send information to other objects.

You connect two objects together with a *patch cord*. Always drag from the outlet of one object to the inlet of the other. When dragging a patch cord, you can be anywhere inside the object box and the inlet will "expand" to indicate that you can release the mouse button to make the connection.

Inlet expanding to accept a patch cord

Normal Objects and User Interface Objects

The most common type of object is the *object box*, which has two lines at its top and bottom.

The action performed by an object box depends on the name you type into it. The name is like a verb, describing what the object will do. The name will either be a single word, such as **makenote**, or a symbol such as **+** (add) or **>** (is greater than).

Typically, objects send information out their outlets in response to information they receive in their inlets. Each inlet and outlet of an object has a different purpose. These are described in detail in the *Objects* section of the **Max Reference Manual**. A brief description of the use of each inlet and outlet is available in the *Assistance* area of the Patcher window, as shown in the next example. Just move the mouse over the inlet, and descriptive text appears in the Assistance area.

Getting Assistance for the right inlet of the **delay** *object*

A few objects lack either inlets or outlets because they receive or send information from some- where outside of the patch that contains them. For example, the **midiout** object has no outlets because it transmits values directly to MIDI.

Other objects are *user interface* objects. They look like buttons, dials, sliders, keyboards, etc., and respond not only to messages received from other objects, but also to clicking and dragging with the mouse. They let you control the program by using the mouse, and also provide a means of displaying numbers and other messages onscreen in a variety of ways. The name of each object is displayed in the assistance area when the mouse rolls over them.

There are some objects, such as **comment**, that do nothing. The comment object lets you put notes to yourself reminding you what you have done, or notes to others telling them how the program works. You can also use comment objects to label user interface items to tell other people how to use your program.

When you've finished making your program, you can save it as a Max document, otherwise known as a *patch*. If you give your patch a single-word filename (that doesn't begin with a number), you can then use it as an object within another Max patch just by typing the filename into an object box.

For a more detailed explanation of Objects and related issues, you can refer to the chapters titled *Objects* and Externals in this manual and the chapter titled *Encapsulation* in the **Max Topics** manual.

Arguments

Often additional words or numbers are included in an object box after the object name to provide initial information to the object or to specify some of its characteristics. The additional words are known as *arguments* to the object.

When you create a metronome object **metro**, for example, you may type in a number argument after the message name to specify how many milliseconds the metronome should wait between ticks.

A few objects have *obligatory* arguments, while others have *optional* arguments. If you forget to type an obligatory argument into an object box, Max will print an error message in the Max window advising you of that fact, and will refuse to create the object. If you choose not to type in an optional argument, Max provides a default value for that argument. The type of arguments that can be given to each object—and their default values—are detailed in the *Objects* section of the **Max Reference Manual**.

Messages

What actually gets sent through the patch cords? A *message* is the information passed from one object to another. A message can be a number, a *list* of numbers separated by spaces, a word (referred to in Max as a *symbol*), or any arbitrary combination of words and numbers. The contents of a message determine its *type*.

The types of Max messages are:

int — When a message consists of nothing but an integer number (such as the message 127), it is understood by Max to be a message of type int. A message could also say int 127.

float — When a message consists of nothing but a number containing a decimal point (such as the messages 3.97 or 3. or .97), it is understood to be a message of type float (short for "floating-point number"). The decimal point lets Max know it is a float. As with ints, it is possible—but not necessary—for a message to say float 3.97.

list — A list of two or more numbers separated by spaces (such as 60 79 1.02 4) is of the type list. The message need not (but may) begin with the word list. Max recognizes a list whenever it sees a message consisting of a number followed by anything. In fact, a list can contain words as well as numbers (as in the message 1 start 768), so long as it *begins* with a number. A list can contain up to 256 items.

bang — The message bang is a special message meaning "do whatever it is you do." For

example, when the **random** object receives the message bang, it sends a randomly chosen number out its outlet.

symbol — A symbol is a word or other non-numeric collection of characters. Many symbols are meaningful commands when received by certain objects. The exact symbols understood by each object are listed in the *Objects* section of the **Max Reference Manual**. For example, the **seq** object, a sequencer for recording MIDI performances, responds to messages received in its inlet such as start, stop, record, delay, and print. All of those messages would be meaningless to an object like * (multiply), which expects only numbers (or bang) in its inlets.

any message — A message can, in fact, consist of *any* combination of words and numbers. Some objects can handle any type of message. Most objects, however, expect to receive one of the above types of message, and will not understand other messages.

When an object receives a message it doesn't understand, it will either ignore the message entirely or will print an error report. In many cases, Max even stops you from making such mistakes while you are editing a program. As you connect an outlet of one object to the inlet of another object, Max does its best to analyze what message will be traveling through the patch cord. Max won't let you connect an outlet to an inlet that won't understand the message. If an inlet is not highlighted when you try to connect a patch cord to it, Max will not make the connection.

Attributes

In version 4.5 and later, a few Max/MSP objects—notably, **mxj** and the pattr family of Max objects **pattr**, **pattrstorage**, **pattrhub**, and **autopattr**)—use *attributes*. Attributes are another way to specify the behavior of Max objects, and are found and used widely in Jitter—an extension to Max and MSP that allows you to work with matrix-based data, including video and OpenGL.

If you already own or use Jitter, you will be familiar with how they work. This section will provide an explanation for users who have not encountered attributes.

You can use attributes to initialize, change, and find out the current values stored in an object, and the attachment of each value to a fixed attribute name means that you don't need to remember the ordering typed-in arguments or what each inlet in a complex object does. Future versions of Max/ MSP will implement attributes for Max/MSP objects much more widely.

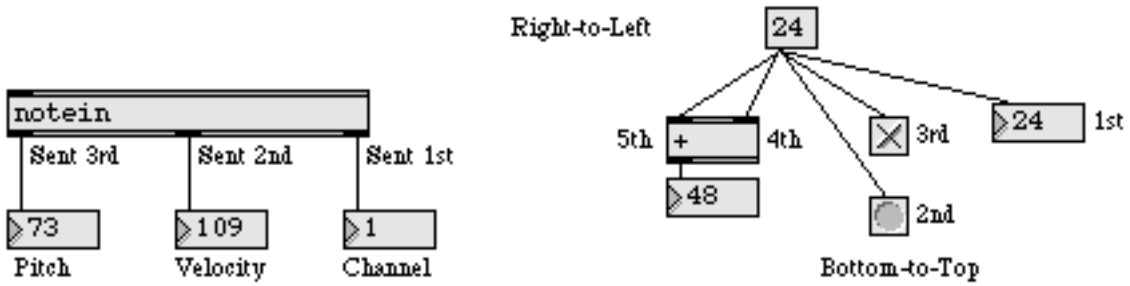
As with arguments, you can type attributes into an object box along with the object's name, or you can set (and retrieve) attributes using Max messages after the object is created. Typed-

You can type in any int as a hexadecimal number. Hexadecimal is a base-16 number system often used in MIDI specifications and other computer-related documentation. Precede a hexadecimal number with 0x (zero plus a small x), as in 0xF0 (same as the decimal number 240).

Message Order

An object often has more than one outlet, and usually sends messages out all outlets "at the same time." Actually, nothing *really* happens at the same time in Max. Things can happen so fast as to seem simultaneous, but it's important to know how Max orders messages.

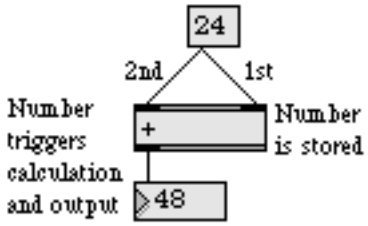
When an object sends several messages "at once", out different outlets, the order is actually *right- to-left*. The rightmost outlet sends its message out first, then the outlet just to the left of that, and so on until the leftmost outlet. This is true of virtually *every* object in Max.

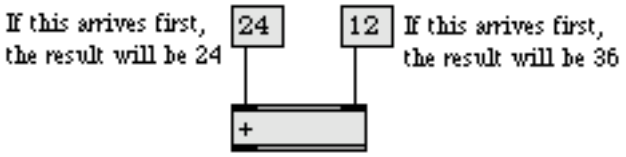


The example on the left illustrates the right-to-left order of message output. In the example on the right, a single outlet has many patch cords connected to it, all leading to different inlets of other objects. In this case, the order that messages are sent is determined by the *right-to-left* screen position of the receiving objects. If two receiving objects are perfectly aligned vertically (neither is further right than the other), the order is *bottom-to-top*; the object that is lower on the screen receives the message first. If a single outlet is connected to two inlets of the same receiving object, the rightmost inlet will receive the message first.

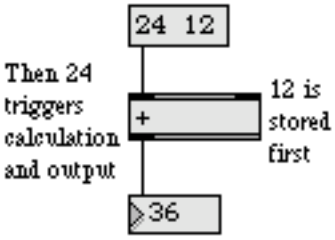
When an object has more than one inlet, it *expects* to receive messages in its inlets in right-to-left order. Generally, an object will store all the messages it receives *until* it receives a message in its leftmost inlet, then it will perform some operation and send messages out its outlets. The vast majority of objects are "triggered" by a message in their leftmost inlet. There are some exceptions to this rule, but the left inlet is the triggering inlet for almost all objects.

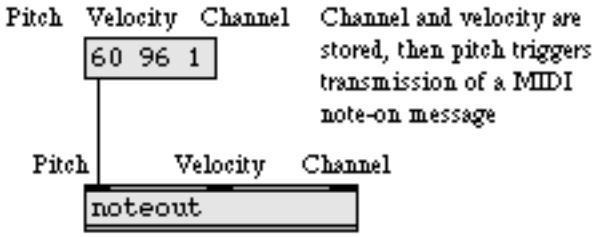
Notice that this makes sense because objects send their messages right-to-left, so the last message an object receives will usually be the one that triggers output.





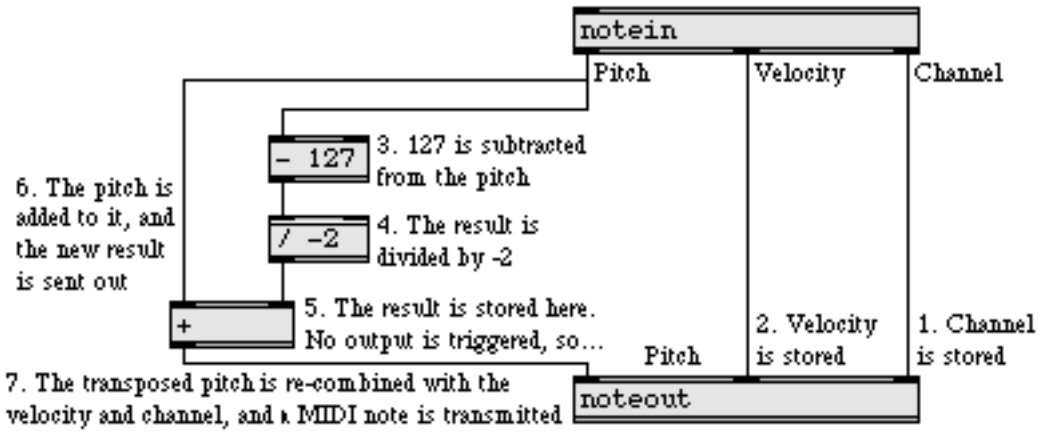
Some objects can receive a list of numbers in the left inlet, and the result will be the same as if the numbers had been received in different inlets in right-to-left order.





Finally, suppose that a message is sent several places from the same outlet, but that message triggers the receiving objects to send *their* messages. What is the order in that case? The answer is that the message will trigger the receiving object, and that object's output will be sent next. The messages will continue triggering other objects until a message is sent that triggers no other action, then Max goes back to the original sending object and sends its next message.

In the example below, note that steps 3, 4, and 5 are all performed (up to the point where no further message is triggered) before the message is sent to the left inlet of the **+** object.

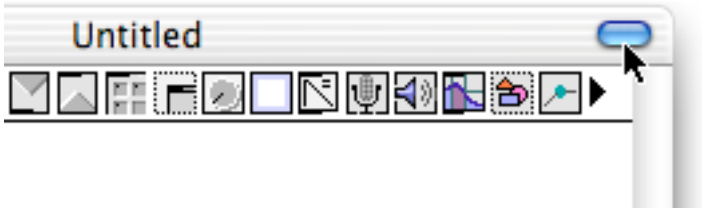


Windows

Max has six main window types: Patcher, Text, Table, Timeline, the Max window, and the New Object List. You can open as many of the first four types of window as you like, but there is only one New Object List and one Max window. Other objects such as env and movie open their own windows.

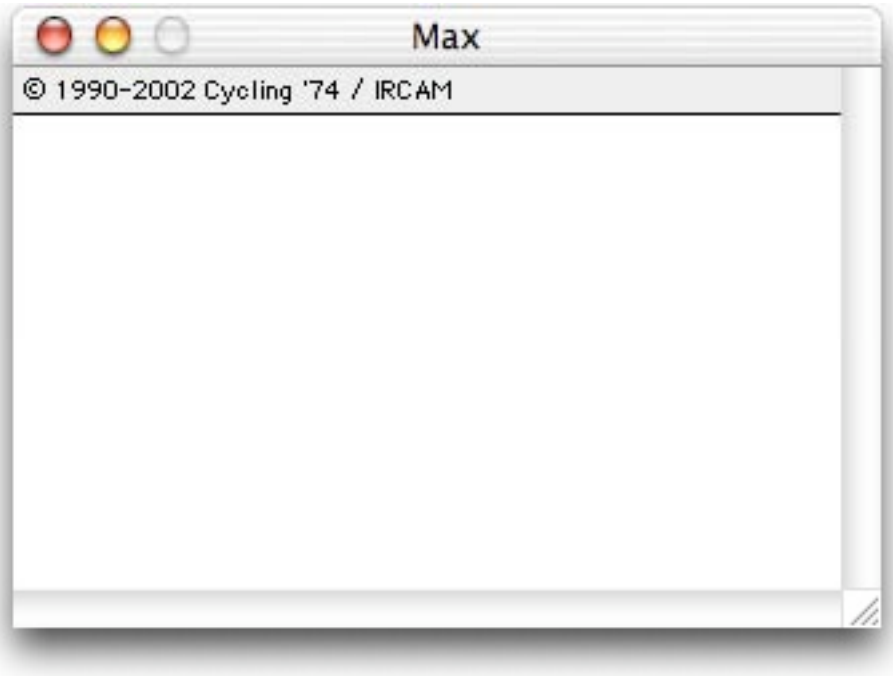
On Windows, each Max window contain the standard icons for minimizing, maximizing and closing the window. Macintosh windows contain the standard icons for closing a window, sending it to the dock, and minimizing/maximizing the window.

Macintosh versions also include a transparent rectangular pill on the right side of the window that can be used to toggle between locked and unlocked Patcher states.



Max Window

When you open the Max application, you'll see the Max window first. This is the place Max prints out messages to the user.



After you launch Max, you may see a few messages from some external objects that were loaded during the startup process.

Any error messages or warnings generated while you are editing or running a program will appear in the Max window. As you're debugging the programs you write, you can have Max print out exactly what is flowing through the patch cords at any given moment, to see if your patch is working properly. For more information on printing in the Max window, see the *Debugging* chapter in the **Max Topics** Manual.

Patcher Window

Once the Max application has loaded, you begin programming by creating a new Patcher window (by choosing Patcher from the **New** submenu of the File menu) or by opening an already existing patch (with the **Open…** command in the File menu).

Programs are "written" in Max by placing objects in a Patcher window and connecting them with patch cords to make patches. Once you have created the patch, you can run it and ensure that it actually does what you want it to do. You can save the patch as a Max document, to run or edit again later.

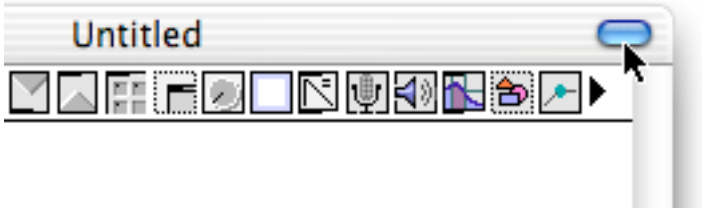
Overview

A patcher window is either locked or unlocked. When it is unlocked, you can edit the patcher by moving objects around, creating new ones, and connecting objects together. When it is locked, you are operating the patcher by clicking on objects (such as sliders) that do things.

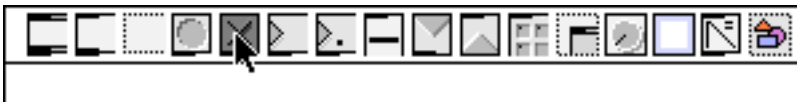
The lock/unlock state of the window is indicated by the presence of the patcher palette at the top of the window. If you see the palette, then the window is unlocked.

There are several other ways you can lock or unlock a patcher.

- Choose **Edit** from the View menu, or type Command-E on Macintosh or Control-E on Windows.
- Command-click on Macintosh or Control-click on Windows on the "white space" in the Patcher window.
- On Macintosh, there is a transparent rectangular pill on the right side of the window that can be used for toggling between locked and unlocked state.



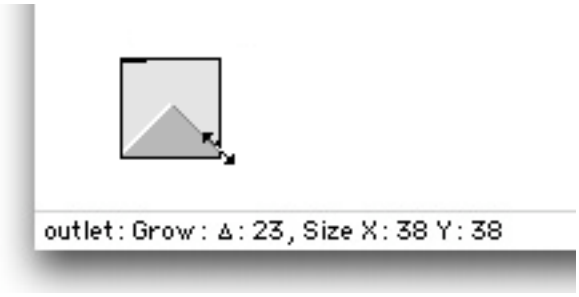
When you create a new Patcher window, it will already be unlocked. To place an object in the window, click on the desired icon in the palette.



The cursor will become the same as the palette icon.

Then click in the Patcher window where you want to place the object.

You can move one or more objects to a new location by selecting and dragging them with the mouse. You can select objects for editing operations such as **Cut**, **Copy**, and **Duplicate**, or if an object box contains text you can change its font or font size by using the Font menu. You can also resize an object by clicking on its grow bar (the tiny rectangle in the lower-right corner of the object). The cursor will change to a two-way or four-way arrow when you are above the grow bar (which is sometimes invisible). As you resize the object, the assistance window will show you the change in the size of the object, and its current dimensions, in pixels.



At any time you can lock the Patcher window and use your patch. When the window is locked, the object palette disappears, and clicking on certain objects now operates them as parts of the user interface. When you have finished editing your patch, and have tested it to see that it runs properly, save it by choosing **Save As...** from the File menu. The next time you open that Max document, it will open locked and ready to use.