Electronic Music and Sound Design
Theory and Practice with Max and MSP • volume 1

Topics

“This book is one of the first courses on electronic sound that explicitly integrates perception, theory, and practice using examples of real-time sound synthesis you can manipulate and experience for yourself. In my opinion, Cipriani and Giri have done a masterful job of allowing experiential and theoretical knowledge to reinforce each other. This book will work either as a textbook or as a vehicle for the independent learner. As a bonus, the book includes a thorough introduction to digital signal processing with Max and serves as a wonderful introduction to the programming concepts in that software. I hope you will take advantage of the excellent Max examples the authors have created. They are simultaneously fun and enlightening, and they sound good enough to use on stage. They are also worth examining as models for your own Max patches, or for extending in new ways. As you will see, the theoretical chapters are the "T" chapters, while practical and experiential knowledge is imparted by the "P" chapters. These chapters alternate, in the form of a ladder, refining the concepts at ever higher levels of sophistication. I want to wish you good luck on this new adventure, and also thank my two Italian friends for creating such a comprehensive resource for learning about digital music—the one I wish existed when I was a student!”

(from the Foreword by David Zicarelli, publisher of Max).

This is the first of a series of three volumes dedicated to digital synthesis and sound design. It is ideal for those who are newcomers to the field, but it will also prove to be an indispensable resource for those who wish to deepen existing skills in sound design, in electronic music, or in Max.

ALESSANDRO CIPRIANI co-authored “Virtual Sound”, a textbook on Csound programming, and was a co-creator of the first online course on sound synthesis available in Europe. His electroacoustic and multimedia compositions have been performed at major festivals and electronic music venues (such as Synthèse Bourges, Venice Biennale and the International Computer Music Conference), and released on CDs and DVDs issued by Computer Music Journal, ICMI, CNI, Everglade, as well as others. He is a tenured professor in electronic music at the Conservatory of Florence, a founding member of Edison South in Rome, and a member of the editorial board of the journal Organised Sound (published by Cambridge University Press). He has given seminars at many European and American universities, including the University of California - Santa Barbara, Stadelschule Academy in Heidelberg, and Accademia di S. Cecilia in Rome.

MAURIZIO GIRI is a professor of composition as well as a teacher of Max/MSP programming techniques at the conservatories of Rome and Frosinone. He is an instrumental, electroacoustic and multimedia composer of music, specializing in digital sound and video processing and computer-assisted composition. He has written computer applications for algorithmic composition and live performance, and has published numerous tutorials on MaxMSP. He was artist in residence at the Cité Internationale des Arts in Paris, and at GRAME in Lyon. He is also an associate member of the Nicos Institute at the École Normale Supérieure in Paris, where he is currently collaborating with other faculty members on a project on the philosophy of sound, entitled “Knowledge of Sound”.

www.contemponet.com
ELECTRONIC MUSIC AND SOUND DESIGN
Theory and Practice with Max and MSP - Vol. 1
NOTICE

This document contains paragraphs that have been added and / or modified in the second edition of “Electronic Music and Sound Design” volume 1 updated to Max 6. The new parts are marked in red.

CONTENTS

Changes pages. 51 - 67 • 1
Changes pages. 71 - 72 • 20
Changes pages. 85 - 86 • 22
Changes page. 90 • 24
Changes page. 94 • 25
Changes page. 97 • 26
Changes page. 109 • 27
Changes page. 118 • 28
Changes page. 151 • 29
Changes pages. 159 - 161 • 30
Changes page. 162 • 33
Changes page. 163 • 34
Changes page. 370 • 35
Changes page. 451 • 36
1.1 FIRST STEPS WITH MAX

In order to follow the procedures outlined in this chapter, you will need to have Max installed on your computer. If you have not done this, or if you have encountered problems, read the document named “How to install and configure Max”, which can be found on the support web page.

Launch Max, and select the New Patcher entry from the File menu (or press Command-n on the Mac or Control-n on Windows). The Patcher Window will appear. The Patcher window is where you connect Max objects to create algorithms. Before proceeding, note the row of icons (named the Patcher Window Toolbar) at the bottom of the window; we will explain the functions associated with these icons during the course of the text. Double click on the inside of the Patcher Window in order to open the Object Explorer, a window that contains a series of categorized icons (as seen in figure 1.1).

The icons represent Max objects that you can use to construct “virtual machines” that embody algorithms for sound synthesis and signal processing. Max objects can be interconnected with each other: information flows (in the form of numbers, digital signals, or other kinds of data) from one object to the next through these connections. Each object is designed to perform a specific operation on the information that it receives, and each object will pass its results.
to any other objects that are connected to it. A collection of objects that are connected together is called a patch (a reference to the old world of analog modular synthesizers that were programmed using physical cable connections called patch cords).

Note that the icons are categorized, and that only two categories, “Basic” and “Audio”, are visible in the figure. You can browse through the remaining categories using the scroll bar on the right side of the window.

Let’s create our first patch. By double clicking on the seventh icon of the Object Explorer, the icon labeled “object” in the “Basic” category, you will cause a new object to appear in the Patcher Window (as seen in figure 1.2).

Fig.1.2 The object box

The object that appears is a generic object called the object box. It is the object that we will use most often when patching. Its function depends upon the name that we give it when we type a string into its interior.

We want to turn this object box into a sine wave oscillator; in order to accomplish this, you will need to type the word “cycle~” into the object box’s interior. Try this now, and notice that as you begin to type, a menu appears that lists all of the objects whose names or whose description contain the characters which you have typed. This very useful feature, shown in figure 1.3, is called auto-completion.

Fig.1.3: the auto-completion menu

2 Note that each category has its icons arranged in alphabetical order
3 The word “string” is computer programming lingo that refers to a combination of letters and numbers, such as “print”, “save”, “astringcontainingthenumber1”, or “subsection22”. 

from “Electronic Music and Sound Design” Vol. 1 by Alessandro Cipriani and Maurizio Giri
© ConTempoNet 2013 - All rights reserved
In the figure, you can see what the auto-completion menu looks like after typing the first two characters of “cycle~” (the ‘c’ and the ‘y’). Notice that the menu is divided into three parts: the first, labeled “Text Completion”, lists objects that begin with the exact characters typed, the second, labeled “Name Matches”, lists objects whose names include the typed characters somewhere in their names, and the third category, “Descriptions and Tags”, lists objects whose description contains the typed characters. Inside the object box itself, immediately following the characters that you have typed, you can see a proposed completion for the object that Max deems most likely to be used (and which is most often a name that you have typed before).

At this point, you can select the object name that you are looking for by clicking on its menu item, or by simply typing the rest of the name; for the purposes of this tutorial, make sure that you select “cycle~” and not “cycle”! With “cycle~” in place, now type a space in order to see how the auto-completion menu changes depending on context. Two new categories, “Arguments” and “Attributes”, appear. Without going into too much detail, the elements that populate these categories represent words that can optionally be typed in after the name of the object. Ignoring these placeholder suggestions for now, continue by typing the number 440 after the space that you have typed. (The space is very important!) After you have done this, click in an empty part of the Patcher Window, and the object box will now resemble figure 1.4.

![cycle~ 440](image)

Fig.1.4 The cycle~ object

The short, darker, lines that appear on the upper and lower edges of the object are the **inlets** and **outlet** for cycle~, respectively, which we will examine shortly. (NB: If the object doesn’t display these, then you have a problem. Look for debugging tips in the FAQ at the end of this section.)

Now we will create another object, gain~, which has an appearance, shown in figure 1.5, that is similar to the faders found on a mixing console. Display the Object Explorer by double-clicking on an empty part of the Patcher Window, and then click on the “gain~” icon, which can be found in the “Audio” category. Alternatively, you could also drag the icon directly into the Patcher Window using the mouse.

When you click on the icon, a new kind of object will appear: instead of an object box, a **new user interface object** (also known as a **ui object**) will be created. Here’s a little tip: if you have a hard time finding an object in the Object Explorer,

---

4 Note the character ‘~’, which is called a **tilde**, that follows the word cycle. The tilde character is almost always the last character in the name of objects used to process streams of digital audio; it is an important naming convention in Max. Since some external objects do not appear in the auto completion menu, you will sometimes need to type (or want to, because it is quicker!) object names directly into the object box. It will be important that you know where to find the tilde on your keyboard, especially if you have a non-English one!.

5 Or alternatively, press <Enter> on the Mac or <Shift-Enter> on Windows.
just create an object box (the same generic object that we just used to create `cycle~`), and then type the name of the object that you want to create (such as `gain~`) into its interior. Once you’ve done this, click outside of the object, and the object box will immediately morph into the ui object whose name you typed.

Drag the newly created `gain~` object to a spot in the Patcher Window underneath the `cycle~` object, and connect the outlet of `cycle~` to the inlet of `gain~` in the following manner: first, bring the mouse pointer near the outlet on the lower edge of the `cycle~` object. A red circle, along with a box containing descriptive text (a feature officially called “object assistance” by Max), should appear. The circle, shown in figure 1.6a, indicates that the outlet is selected, and by clicking and dragging the mouse towards the `gain~` object while holding the mouse button down, a yellow-and-black striped patch cord can be made to appear. If you drag the end of this patch cord towards the inlet of the `gain~` object (the small dark patch on its upper edge), you will see another red circle (and more descriptive text) appear, as shown in figure 1.6b. This second circle indicates that you have located an inlet that is compatible – at this point, release the mouse button, and you will have connected the two objects!

The `gain~` object has two inlets on its upper edge (which are really quite difficult to see as distinct, since they are crammed together and poorly distinguished from each other). The left inlet, to which we connected the `cycle~` object, is designed to receive audio signals from a generator such as `cycle~`, and the right inlet is designed to receive numeric values which, for the time being, we are not interested in.6 It is impossible to make buggy connections by routing the digital signal outlet to the numeric inlet, because Max will not allow incompatible connections such as this to be made.

---

6 For the curious, this numeric inlet represents the “slew rate” for the fader handle, or the time interval in milliseconds used for interpolating fader positions while it is being moved.
Addendum 2nd edition “Electronic Music and Sound Design” volume 1

At this point in our tutorial, we need to create a graphical object called an **ezdac~**, which appears as a tiny loudspeaker in the “Audio” category of the Object Explorer (as seen in figure 1.7).

**Fig. 1.6 Connecting two objects**

**Fig. 1.7 The ezdac~ object**
We place the `ezdac~` under the `gain~` object and connect its left outlet to the two inlets of the `ezdac~` (as shown in figure 1.8).

![Fig.1.8 Our first patch](image)

Watch out! The `gain~` object has two outlets, which are again poorly distinguished from each other. Because of this, you should verify that you’ve used the left outlet for making both connections. The best way to ascertain that you’ve used the correct outlet is to read the object assistance text that appears when you make the connection, and to make sure that it contains the text `gain~: (signal) Scaled Output`. If one of the two cables were to be grey rather than yellow-and-black striped as they appear in the figure above, this would indicate the you’ve mistakenly used the right outlet, in which case you would need to cancel the patch cord by selecting the cord with a click (causing it to appear fatter) and pressing the `<Delete>` key (which is the same key that you press to delete text). The cord would disappear, and you could then reconnect the objects, using the correct inlet/outlet pair.

Now would be a good time to save your new patch to disk, keeping this warning in mind: DON’T EVER save your patches to a file that shares a name with a pre-existing Max object! For example, don’t call this patch `cycle~` (or `cycle` without the tilde, for that matter!) Doing this would be a recipe for confusing Max, and for causing unexpected results the first time that you tried to reload your patch. Given that it is impossible to remember all of the names for Max objects, a good technique for avoiding the use of object names, and therefore averting the danger of a confusing name, is to give your files a name composed of more than one word: “test oscillator”, for example, or “cycle~ object test”, or any other combination. No Max object possesses a name composed of more than one word.

Don’t forget this advice! A large percentage of the problems encountered by Max beginners are related to saving files that share a name with some internal object. We will return to this topic and explain the logic behind it in the interlude that follows this chapter.

Good! We’ve finished implementing our first patch, and we are ready to make it run. It lacks one more touch, however: up till now we’ve been in `edit mode`, in
which we assemble patches by inserting, moving and connecting objects together, and now we need to make the transition into **performance mode**, where we will be able to hear and test our patch. To do this, click on the small padlock that appears at the bottom left of the Patcher Window, or else press <Command-e/Control-e>. Once we are in performance mode, the padlock at the bottom left will appear to be locked. (If it appears to be open, you are still in edit mode!)

Having switched to performance mode, click on the **ezdac~** object (the small loudspeaker), and then slowly raise the level of the **gain~** slider. You should hear a sound, pitched at A above middle C. By clicking repeatedly on the **ezdac~** loudspeaker, it is possible to alternately disable and enable sound output for the patch. (If you are not hearing any sound at all, you should try consulting the FAQ at the end of this section.)

Now that we've built a patch and seen it work, let's revisit what we did: The **cycle~** object is an oscillator (a sound generator that produces a periodic waveform, in this case, a sine wave), and the number ‘440’ that we typed into its interior indicates the frequency that we want it to produce; in this case, we specified that the sine wave should repeat itself 440 times per second. In more formal terms, **cycle~** was the name of the object, and 440 was the **argument**; a value used by the object to specify its operation. In this case, the argument 440 caused **cycle~** to produce a tone at 440 Hz.

The signal outlet of the **cycle~** object was connected to the inlet of the **gain~** object, which caused the signal generated by **cycle~** to be passed on to the **gain~** object, where it could be modified, as we have seen, by moving the volume fader. The modified signal was then passed on to the **ezdac~** object (the little loudspeaker), which attenuated the signal and routed it to the sound driver, which managed a digital-to-analog conversion that transformed the numeric representation of the sound into an audio wave that you could hear on your headphones or speakers. (This conversion, by the way, is where **ezdac~** got its name; it is a quasi-acronym for EaZy Digital-to-Analog Converter.)

Let's now broaden the function of this patch. Let's make it possible to actually see what is happening in addition to hearing the sound. Save the current patch (which you will need again in the next section) into an appropriate folder, for example “My Patches”, and close the Patcher Window. If you haven’t already downloaded and unpacked the “Max Chapter Materials Vol 1” archive that can be found on the support web page, do this now.

Open the file **01_01.maxpat**, which you will find in the “Max Chapter Materials Vol 1/Max Patches Vol 1/Chapter 01 Patch” folder (see figure 1.9).

---

7 Alternatively, you can move back and forth between edit and performance modes by holding down the <Command> key (on the Mac) or the <Control> key (on Windows) and clicking the left mouse button on an empty part of the Patcher Window.

8 In this case, it is actually a cosine wave, as we will see in the next chapter.

9 All of these concepts were laid out in Theory Section 1.2.

10 A sound driver is a piece of system software that enables programs such as Max to manage incoming as well as outbound audio signals.
When this file opens, you will see that we have added new objects to the original patch. The new objects on the left, in which you can see numerical values, are called *number~* objects, and they show, via the value that they display, a snapshot of the signal that they are receiving. The larger rectangular objects on the right are called *scope~* objects, which act as oscilloscopes on which it is possible to view an incoming signal as a graphical waveform. The object [p gain_to_amp] and the object that is connected to its outlet (which is called a *flonum* or a float number box) are used to view exactly how much gain~ is being applied to amplify or attenuate the signal.

Once again, start the patch by clicking on the *ezdac~*, and observe the numbers changing in the *number~* object high on the left. These numbers are being produced by the *cycle~* object and, if we observe them for a bit, we can see that their values, both positive and negative, fall between 1 and -1. On the upper right side we can see the *scope~* object displaying the same numbers in graphical form; the upper half of its panel corresponds to positive numbers, and the lower half to negative numbers. In the *scope~* panel, hundreds of different elements are shown, visualized as a sequence of points, rather than the single number shown by the *number~* object. The points fall very close to each other on the screen, and so they appear as a solid curve. The elements that they represent, the numbers themselves, are called *samples* in the terminology of digital music. And the line made from these sample values, as they undulate high and low across the oscilloscope panel, is precisely the sinusoidal wave produced by the *cycle~* object.

---

11 The objects *number~* and *scope~* can be found, like the first objects, on the Object Explorer. If you want to create new *number~* and *scope~* objects in your patch and don’t want to use the Object Explorer, you can always use the trick that we demonstrated above: take an object box and type the name of the desired object into its interior. We will see later how to make searching for icons easier.
The patch also contains a second `number-` and a second `scope-`, each connected to the `gain-` object. These objects should be displaying the number 0 and a flat line (which is, of course, a sequence of zeros), because the volume fader is at its lowest setting, resulting in a volume of 0. If we move the `gain-` fader upwards, we should see `number-` begin to display values that start out very small and gradually grow larger as the volume rises, and at the same time, the flat line of the lower `scope-` should begin its undulation and assume the same look as the other `scope-`. We can infer from this that `gain-` is controlling the amplitude of the signal— the more we raise the fader, the greater the amplitude of the oscillations becomes. If we go too far, and raise the value of the `gain-` fader to be close to its maximum setting, we see `number-` begin to exceed the amplitude limits of 1 and -1, and the waveform on the oscilloscope becomes clipped. More important than these visual clues, you should be able to actually hear the sound change, as it becomes distorted.

We can draw some conclusions from what we’ve seen:

1) The `cycle-` object produces a sequence of digital values that follow the pattern of a (co)sine wave.

2) The numerical limits for samples in this sine wave are 1 and -1. The actual sequence that these values follow can be seen on the upper `scope-`, which shows the waveform at its maximum amplitude, above which the quality of the sound would be distorted.

3) The `gain-` object modifies the amplitude of the sine wave, causing the sample values at its outlet to be different than the corresponding sample values received on its inlet. How does it do this? By multiplying the values that it receives by a quantity that depends upon the position of the fader. When the fader is in its lowest position, the signal is multiplied by 0, and the result is a stream of zeros (because any number multiplied by 0 is 0). One can see that as we raise the fader, the multiplication factor rises. If, for example, we move it to 0.5, the amplitudes of the samples that enter the `gain-` object are diminished (because multiplying a number by 0.5 is the same as dividing by 2).12

As we move it to 1.0 (which is about 3/4 of the way up the fader), the sample values entering the object are identical to those leaving. Finally, if we raise the fader all of the way, the most extreme of the sample values will exceed the limits of 1 and -1, although these samples will then be brought back into line during

---

12 To adjust the fader to a position that corresponds to a multiplication factor of 0.5, watch the number box that is connected to the `p gain_to_amp` object, which is set up precisely for the purpose of displaying the value of the multiplication factor. The fader is actually divided into logarithmic increments, and using a formula that we’ll explain later, the `p gain_to_amp` object converts these fader positions (which can be read from one of the outlets of the `gain-` object) into an amplitude level. We will return to this topic in Interlude A, which follows this chapter. One thing to note is that when the multiplication factor is near 0.5, the sine wave takes up about half of the oscilloscope screen.
the digital-to-analog conversion. When that happens, however, the waveform will no longer be a sine wave. Instead, it will be clipped (as we can see in the lower oscilloscope). Sample values that fall outside of the -1 to 1 range are actually simply reassigned the maximum possible amplitude during conversion, and the distorted sound that we hear reflects the resulting truncated waveform.

We have continued the exploration of the concepts of frequency, amplitude, and waveform, which we began in Section 1.2 of the theory chapter. Let’s recap some practical aspects of these basic concepts:

- Amplitude is the physical parameter that corresponds to the intensity of a sound; it is the parameter that controls the perception of whether a given sonic event is forte or piano. In Max, the absolute value of amplitude (that is, the value of the parameter, independent of its sign) always lies between 0 and a maximum of 1.

- Frequency is the physical parameter that relates to pitch; it is the parameter that controls the perception of whether a given sonic event is high or low in pitch. The values are expressed in hertz (Hz), and we need to keep in mind that sounds audible to humans fall in the range between 20 and around 20,000 Hz.

- Waveform is a fundamental element of timbre, which we define as the overall quality of a sound. Timbre enables us to perceive, for example, the difference between the middle C played on a guitar and the same note played on a saxophone. We have seen and heard the timbre of the sine wave produced by `cycle~`.

**FAQ (Frequently Asked Questions)**

In this section, we will try to give answers to some of the more common problems that are encountered by new users of Max. You should probably read these carefully even if you have not experienced a problem, since they contain information that you will use in the following sections of the book.

**Question:** I created an object called “cycle~440”, as instructed in this section, but the object has no inlets or outlets. What went wrong?

**Answer:** Be sure that you typed a space between “cycle~” and “440”, because the first is the name of the object, while the second is an argument, which in this case represents the frequency of the sound. If the two words are run together, Max will search for a non-existent object named “cycle~440”, and when nothing is found, Max will have no inlet or outlet information with which to work.

**Q:** Very good. Why, then, didn’t Max give me an error message?

**A:** There is an error message, which can be found in the Max Window, which is a window that the program uses to communicate with you. If you cannot see
this window, press <Command-m/Control-m> to bring it up. In the window, you will probably find a message such as “cycle~440: No such object”. If you double-click on the error message, the object that caused it (in this case, the bogus “cycle~440” object) will be highlighted in the Patcher Window.

Q: I inserted a space between “cycle~” and “440”, but the object has no inlets or outlets just the same!

A: There is a more subtle error that often turns up for some new users when using objects that have a tilde (‘~’) at the end of their name. If you have a keyboard on which there is no tilde key, and so you need to press a combination of keys in order to produce the character (for example, <Alt-5> on some Macs), you may have continued pressing one of the modifier keys when typing the space (for example, <Alt-Space> on the Mac). The resulting combination is not recognized by Max, and Max is not able to separate the object name from its argument. Delete the space and re-insert it, avoiding pressing modifier keys at the same time.

Q: There is no sound.
A: Have you clicked on the ezdac~ object (which is the loudspeaker icon)? Have you raised the volume fader above zero? Are you sure that sound on your computer is not muted, or that you are able to produce sound by using a program other than Max? Have you checked the Audio Status window (which you can find in the Options menu), to see that the correct sound driver is selected? If you still have no success, re-read the “How to Install and Configure Max” document, which can be found on the support web page.

A COMPACT “SURVIVAL MANUAL” FOR MAX

In this section, we will give some essential information for easily navigating the Max environment.

BASIC KEYBOARD COMMANDS

Mac <Command-n> or Windows <Control-n> will create a new Patcher Window, which is the workspace for implementing patches.

Mac <Command-e> or Windows <Control-e> will toggle between edit mode and performance mode in the Patcher Window. In edit mode, we assemble patches by creating objects using the Object Explorer; in performance mode we can activate patches and interact with their graphical user interface objects (such as float number boxes or the gain~ object).

Mac <Command-m> or Windows <Control-m> will call up the Max Window if it is not already visible. The Max Window is a window used by the Max environment to communicate, using brief text messages, with its user. We will learn more about this window shortly.
Instead of using the Object Explorer, it is possible to create some common objects by typing simple one-character commands, without having to use <Command> or <Control> keys. The character ‘n’, for example, will create (when your Patcher is in edit mode) an empty object box at the position of the mouse pointer. There are other keys that also enable object creation: in edit mode, try positioning the mouse pointer and pressing ‘f’, ‘i’, ‘t’, and ‘b’, and observe the various objects (which probably mean nothing to you at the moment!) that are created. We will use these objects frequently in the course of the following chapters.

SELECTION, DELETION, AND COPYING
To delete a patch cord or an object, you need to insure that you are in edit mode, select the patch cord or object with the mouse, and then press the <Delete> key (which may also be referred to as the <Backspace> key). We can select multiple objects at a time by clicking on a blank spot in the Patcher Window and dragging a rectangle that “touches” the objects to be selected (see fig 1.10).

At this point, if we move one of the selected objects, all of them will move; likewise, if we press the <Delete> key, all of the selected objects will vanish. Using this selection technique, objects will be selected but not patch cords. If we need to select the patch cords at the same time we are selecting objects (for example, to delete them) we need to hold the <Alt> key down while dragging the selection rectangle, and make sure that we “touch” the cables that interest us (as seen in figure 1.11).

With the <Alt> key held, we can also copy an object. Simply click-and-drag to tear off a copy. If you first select multiple objects and then drag them using <Alt-Click>, you can copy them all (as in figure 1.12).

13 Check to make sure the padlock at the bottom left of the window containing the patch is open.
Fig. 1.12 Copying multiple objects at once

If you make a mistake (for example, you delete one object rather than another), you can undo the action by selecting the Undo command from the Edit menu. Repeatedly selecting Undo can cancel an entire sequence of commands and return the patch to the state that preceded those actions. If, after undoing one or more actions, you decide that there wasn’t actually an error (for example, you wanted to delete the object after all), you can restore actions by using the Redo command, which is also found in the Edit menu. The command equivalents of undo and redo are <Command-z> and <Shift-Command-z> on the Mac and <Control-z> and <Shift-Control-z> on Windows.\footnote{<Shift> is the modifier key used to type capital letters.}

HELP

This book is self-contained: within it you will find all that you need for understanding and using the patches with which we illustrate Max synthesis and signal processing techniques. But the Max environment also provides extensive illustrative materials that can be accessed by selecting the Max Help item from the Help menu. When you select this item, you will obtain the window shown in figure 1.13 (which may be quite different looking in different versions of Max).
In the main part of this window are links to many resources that you can use to get acquainted with Max, while on the right side there are three tabbed panels: the first contains a categorized list of all of the objects in Max, the second has a list of tutorials that explain various aspects of Max programming through brief illustrations of synthesis and sound processing techniques, and in the third lists “Vignettes”, which are short, focused, documents that highlight specific Max features. You should definitely take some time to browse Max with this help system as your guide! We recommend that you start with the section entitled “Learn About the Documentation”, which will teach you how to move between sections and how to find topics that are relevant to your interests. This is not absolutely necessary for the comprehension of this book, but it can be very valuable.

There are also help patches for all of the individual objects found in Max. When you are in edit mode, <Alt-Click> on an object (without dragging), and a help patch relevant to the clicked object will open. This patch will be fully functional, and will summarize the principal characteristics of the object. <Alt-Click> while in edit mode on the cycle~ object, for example, will bring up the help patch seen in figure 1.14 (which again may be quite different in different versions of Max).

Fig. 1.14 A help patch

Help patches all share a very specific structure: they are divided into panes that can be displayed by clicking on tabs that are visible in the upper left part of the
window: each individual pane explains some specific characteristics or behaviors for the object in question. The number of tabs and the names on these tabs will vary from object to object, with the exception of the first and last tabs, which are shared by all objects and are labeled consistently. The first of these shared tabs, labeled “basic”, highlights the fundamental features of the object being examined, while the last shared tab, labeled with a question mark, contains a menu whose first item, “Open Reference”, can be used to call up a detailed page from the reference manual, and whose succeeding menu items lead to help patches for related objects, to objects that are similar to the object in question, and finally, to tutorials illustrating the use of the object. Whether or not you read the deeper explanatory material contained in a help patch such as this one, we still recommend that you take a peek! They are working patches that can teach a great deal about the practical techniques and specialized vocabulary used in Max.

Another useful source of information is the Clue Window, which can be called up via the Window menu. This window, when viewed using the default (assumed) color scheme and settings,\(^{15}\) appears as a small yellow window that floats above other windows and that displays information relating to whatever lies beneath the mouse pointer. Try activating it and then moving the mouse pointer over the various elements of a patch, or over the icons found at the bottom of a Patcher Window, or, finally, over any of the numerous menu items. The clue window will display a short description for all of these different items.

The comprehensive help system is undoubtedly one of the strongest points of the Max environment. We have already encountered other aspects of it: remember the “object assistance” text, which we see every time that we work with patch cords. Object assistance dynamically describes messages that can be sent to or received from an object’s inlets and outlets. Recall that to see it, we need only hover with the mouse over an object’s inlet or outlet when in edit mode (as in figures 1.6a and 1.6b). Another very useful resource in the same vein is the Quickref menu. Open 01_01.maxpat once again, and put the patch into edit mode by clicking on the lock at the bottom left of the Patcher Window. Now hover with the mouse pointer over the left inlet of cyc1e~ so that the red circle and object assistance appear. Right-clicking while inside the red circle, and then holding the mouse button down, will cause the Quickref menu to appear (as shown in figure 1.15).

\(^{15}\) Default colors and settings can be easily modified by Max users, resulting in alternative an look and feel for the program.
In the Quickref menu there are three categories of elements: Messages, Actions, and Attributes. We can use Actions to open the help patch for an object or related ref pages. Higher up in the menu, Messages correspond to the messages that a given object is able to “understand” and use. By selecting a message from the list, it is possible to create a new object that is automatically connected to the target object (in this case, cycle~). Click, for example, on the int [int] item as shown in figure 1.15, and a new object will appear, already connected to cycle~, as in figure 1.16.

If you now put the Patcher into performance mode (by closing the lock in the lower left with <Click>), and make vertical sliding strokes of the mouse over the new object while holding the mouse button down, you will see the numbers change in the object’s UI. This object’s purpose is, in fact, to store and enter integer numbers; it is named number, or number box. Numbers that we generate by sliding the mouse over this object are sent to cycle~ to modify...
its frequency. (You will note, however, that the argument of 440 that you can see inside of the cycle~ object doesn’t change. Instead, the frequency is over-ridden invisibly by the newly transmitted values.) Run the patch by clicking on the loudspeaker icon and raising the fader to about 3/4 of its range, and then set the number in the number box to values between 500 and 1000 by sliding the mouse. You will hear the oscillator change pitch, driven by the frequencies being output by the number box. We will speak at much greater length about the number box in following sections of this chapter.

THE OBJECT EXPLORER
As we’ve already said, you can use the Object Explorer to create the Max objects that you need to create patches. To summon the Object Explorer, all you need to do is double click on an empty spot within a Patcher Window that is in edit mode, or else to type the single character ‘p’. There is another alternative, which is to activate Max’ sidebar by clicking on the Patcher Window Toolbar’s last icon, as shown in figure 1.17. Clicking on this icon will cause a tabbed panel to appear, attached to the right side of the Patcher Window in which four tabs enable you to move between the Object Explorer, the Max window, a page from the reference manual, and the inspector window (which we will discuss later). Right now, we will take a moment to expand on the Object Explorer.

![fig.1.17: examining the Object Explorer within the sidebar.](image)

When you click on an icon in the Object Explorer, you will see a brief description of that object appear in the lower part of the window. Also, at the top of the window, you will find a series of buttons in a region labeled “Show”, which can be used to restrict the kinds of objects being displayed. There are six buttons in this region, the first of which, labeled All, can be used to show all objects, while the one labeled UI Objects will select only interface objects, the one labeled Max will show Max objects only, MSP will show MSP objects, Jitter, Jitter, and so on.
objects (which we will discuss in a future volume), and finally Patchers will show
custom objects created in Max that are available on your computer as patchers
(and which we will discuss in a later section). Below the buttons is a menu labe-
led “Category” that will help you to further restrict the objects shown through
categorization. Finally, there is a text field that you can use to search for objects.
Type part of a word or a name into this field in order to find and display related
objects. (The cycle~ object can be found, for example, by typing the word
“oscillator” into the search field; remember to have the All button selected in
order to issue the widest search.)

SOME ORDER FOR A CHAOTIC WORLD
You probably noticed that some of the patch cords in patch 01_01.maxpat
(figure 1.9) had angles in them which divided their connections into tidy seg-
ments. How was this done? The answer is simple: by selecting **Segmented
Patch Cords** from the **Options** menu.
If you use this option, the procedure for connecting two objects will be slightly
different than what we’ve seen already: begin connecting with a click on the
outlet that will form the source of the connection, and then pull the patch cord
out from the object **without holding the mouse button down**. (The cable will
remain hooked to the mouse pointer.) Segments can be created at any time
with a click of the mouse at the point at which you want to change direction;
for every click, there is a new segment. The final click should be made on the
inlet of the object to which you wish to connect.
If you make an error and want to release the cable without connecting, you
need to <Command-Click> on the Mac or <Control-Click> on Windows, or else
hit the escape key (<Esc>).

If you select some objects that are lined up roughly horizontally, and then press
<Command-y/Control-Shift-a>, the objects will become perfectly aligned. The
same command works to create vertical columns of objects one over the other.
(The two scope~ objects and the two number~ objects in 01_01.maxpat were
aligned in this way.) Objects can also easily be aligned using the **Snap to Object
function** (which is active by default). Every time that you position an object in
a patch, this function will align that object with other nearby objects. Another
related function is **Distribute**, which can be found in the **Arrange** menu; this
function makes it possible to distribute a group of selected objects at equal
intervals horizontally or vertically.

A complex patch can be very crowded, with dozens of objects and patch
cords stretched between them. In this case, it is possible to render a set of the
objects and cords invisible when in performance mode, but still visible while in edit mode, so that they can do their work behind the scenes. To hide one or more objects and patch cords, select them while in edit mode and press <Command-k/Control-k>. When the Patcher is put into performance mode, the objects will vanish. To reverse this action, and make the objects reappear during performance mode, use <Command-l/Control-l> (lower key “el”) in edit mode. By selecting multiple objects, it is possible to hide multiple objects at once using these commands. An alternative to the keystroke commands is found in the Object menu, where Hide on Lock and Show on Lock can be found. Play with 01_01.maxpat, making various objects and patch cords in the patch appear and disappear.

There is yet a still more efficient way to put order into patches, which is to use a feature called presentation mode. We will explain this mode once we find ourselves building a more complex patch.

**EXERCISE**

Create a new Patcher Window and attempt to reproduce the patch contained in the file 01_01.maxpat.

Pay careful attention that you don’t confuse the number~ object with the number box! If you don’t succeed in finding scope~ and number~ in the Object Explorer, remember that you can always take an object box, type the name of the object into its interior, and morph the object box into the graphic object that you seek. Note that the displayed waveform on the oscilloscope created by you is different from that in the original file. We will see why in the following section.
Pp. 71-72, the text from Figure 1.23A to fig. 1.23b is replaced as follows:

![Fig. 1.23 The inspector for the scope~ object](image)

What we see is the inspector for the `scope~` object, a window through which it is possible to configure various properties of an object. All objects in Max have an inspector, and we will see many of them in the course of this book. The properties of an object (called attributes in Max) are subdivided into categories in the inspector, and these categories can vary from object to object. The inspector for a `scope~` object has the categories “Appearance”, “Behavior”, “Color”, “Description”, “Name”, and “Value”. We will look at the use of some of these categories during the course of this book, but for now, let’s look only at the attributes found in the “Value” category for `scope~`

The first attributes that we see are `Buffer Size` and `Calcount - samples per pixel` (...
By manipulating the "Color" category attributes we can modify colors for elements such as the signal trace or the background (…).
Pp. 85-86, the text from figure 1.38 to figure 1.40 is changed as follows:

Fig. 1.38: Invoking the inspector by using its patcher window icon

You can also activate the sidebar (see figure 1.17) and select the Inspector tab, as shown in figure 1.39.

Fig. 1.39: the inspector as it appears within the sidebar

In figure 1.40 we can see the collection of “Value” attributes that are shown for the function object in the inspector.
First of all, take a glance at the eighth item, *Lo and Hi Display Range* (…).
Page 90, the first line is replaced by the following text:

It is also possible to use the function object to manage a curve~ object. To do this, you need to open the inspector for a function object and change the value of its Mode attribute (found in the “function” attribute category) to “curve”. Once you have done this, when the patch is in run mode you can change the curve parameters used for any given segment by holding the <Alt> key down and grabbing and dragging the desired segment with the mouse.

Let’s examine one aspect of curve~ more closely: (...)
Page 94, after figure 1.49 the following text is added:

Fig. 1.49 The *kslider* object in the Object Explorer

Note that this object is found in the “Sliders” category of the Object Explorer, which can be difficult to reach if all of the objects in Max are being displayed. We recommend that you work with the “UI Objects” button pushed in the upper region of the Object Explorer in order to restrict the objects being shown to user interface objects only.
Page 97, the sub-section “PROVIDING INFORMATION FOR USERS” is changed as follows:

If you compare the patches shown in figures 1.51 and 1.52, you will note that in the second patch we have removed the comments near the number boxes, but that if you pass the mouse over these number boxes in performance mode, you will see the same commentary text appear in a small yellow floating square. This is done by using the **hint** feature, which can be accessed from the inspector. In edit mode, call up the inspector for one of the number boxes, and find its Hint attribute in the “Description” category. Edit the text field in the box next to the attribute name in order to provide the text that will be shown in the yellow rectangle. If, on the other hand, you’d like to provide a comment that will appear in the Clue Window when the mouse passes over your object, you can type this message into the item named “Annotation” on the same inspector panel.
If you try, while in edit mode, to select all of the objects lying above a panel using the technique illustrated earlier in figure 1.10, you will notice that it is nearly impossible to not also select the panel itself. This happens because the panel, although located at a lower level, is still an object that can be dragged, modified, or deleted like the others. Another inconvenience is that patch cords are hidden by the panel object when you go into performance mode. To solve this problem, we need to push the panel object to an even lower level – the so-called “background level” which is always present behind the “normal” foreground. To do this, select the panel and then invoke the “Include in Background” item in the Arrange menu. After this, toggling the “Lock Background” item from the View menu or pressing <Command-Alt-l/Control-Alt-l> will “freeze” all of the background objects, making them unselectable in any mode. To unfreeze the objects, toggle “Lock Background” or press <Command-Alt-l/Control-Alt-l> again, which will uncheck the option in the View menu and leave the objects accessible once again.

43 Remember that this technique for multi-selecting objects consists of clicking and dragging the selection rectangle across all objects to be selected.
Page 118, the first two items are changed as follows:

**Search for objects possessing analogous or complementary functions**
In edit mode, select object + <Alt-click> on the object to obtain its help patch. In the upper left of the help patch there is a row of tabs; the last of these, labeled with a question mark, will lead to a menu from which it is possible to open the help patch for objects that either similar to or that are often used together with the selected object.

**View the reference page from the manual for an object**
In edit mode, select object + <Alt-click> on the object to obtain its help patch. In the upper left of the help patch there is a row of tabs; the last of these, labeled with a question mark, will lead to a menu, and the first item in this menu, “Open Reference”, will open a page from the reference manual that contains detailed information about all of the characteristics of the selected object.
Page 151, the first paragraph is changed as follows:

To load the file for an abstraction once you’ve got it open, you can click on the second icon in the Patcher Window Toolbar at the bottom of the relevant Patcher Window. (The one immediately after the padlock icon.) This button will open a menu from which you can select an item labeled “Open Original”, which is exactly what you are trying to do. Once the abstraction is modified and saved to disk, all of the patches that contain it will immediately use the new version.
Pp. 159-161, the subparagraphs “THE UNPACK OBJECT” and “THE PACK OBJECT” are replaced as follows:

THE UNPACK AND UNJOIN OBJECTS

A list, as you know, is a message composed of multiple elements, which can be unpacked using the **unpack** object, which needs as many arguments as there are elements in the list. Each argument serves to specify the type of an element. In figure IA.47a, we see an example (which you should build, as usual).

Fig. IA.47a The **unpack** object

In this example, a message box containing a list of four elements is connected to an **unpack** object which has four arguments: the first two are floating point numbers, the third is an integer, and the fourth is a string or a symbol (represented by the letter ‘s’). Clicking on the message box, the four elements are separated and sent to number and message boxes connected to **unpack**. (As we saw in the preceding section, to display a message sent to a message box, we must send it to the right inlet.)

Another object that breaks lists into pieces is the **unjoin** object, which takes a numeric argument that specifies how many elements it should extract from any list that arrives on its inlet. (See fig. IA 47b.)

Fig. IA.47b: the **unjoin** object

In this case, it isn’t necessary to specify the types of the list items being unpacked (whether they are numbers, integers, floats, or strings). As you can see, the argument for the object in the example is 4, which signifies that the object will be able to break a list arriving on its inlet into four parts. But notice that the **unjoin** object in the figure has five outlets rather than four; the final outlet is there to handle the possibility that there might be more elements in the list than specified by the argument. If, for example, we send the list [1 2 3 4 5 6 7] to the object in the figure, the first four list elements will be unpacked and passed to the first four outlets, while the remaining elements will be passed as a list [5 6 7] to the fifth and final outlet. (Build this patch to try this out.)
THE PACK AND JOIN OBJECTS

A series of independent elements can be packed together into a list by using the **pack** object, which takes as many arguments as there are elements in the list. Each argument specifies the type of the element and its initial value. In figure IA.48a, we see an example (which you should recreate!):

![Fig. IA.48a The pack object](image)

Here, three number boxes are connected to a **pack** object that has three arguments: a floating point number, an integer, and another floating point number. The object gathers the three numbers together into a list, which we can inspect by sending it to the right inlet of a message box. If you try modifying the number boxes, note that the message box updates only when the number on the left is modified, exactly like most other Max objects. Note also that since the second argument of **pack** is an integer, if you send a floating point number to the second inlet, it will be truncated and transformed into an integer. Referring to figure IA.48 we see that sending the second inlet the number 1.5, the second element of the list is the integer 1.

A second object that can be used to construct lists from individual elements is the **join** object, which takes a numeric argument that specifies the number of inlets to make available to form a list. (See fig. IA.48b.)

![fig. IA.48b: the join object](image)

In this figure, which we encourage you to build yourself, you will see a **join** object being used to create a list that contains three elements. Each inlet can receive messages of any type: integers, floats, strings, or more interestingly, lists. What this means is that it is possible to build lists with more elements than were originally specified by the numeric argument of the **join** object. Try this by building the example patch shown in figure IA.48c.

![fig. IA.48c: sending lists to a join object](image)
In this example, we have formed a list containing eight elements by sending lists on each of its three inlets. Based on this capability, the \texttt{join} object might seem to be more flexible than \texttt{pack}, but in some cases, \texttt{pack} is indispensable. For example, the arguments of \texttt{pack} actually specify the initial value of an element in addition to its type, a feature that is unavailable in \texttt{join}.

Figure IA.49 demonstrates this by adding a fourth argument to our \texttt{pack} object, without necessitating another number box for updating its value.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig IA.49.png}
\caption{Passing an argument as an element of a list}
\end{figure}

The list produced is made up of four elements, and the last element maintains the value that we gave it as an argument (100 in this case). Thanks to this behavior, we can, for example, send lists that contain fixed values by setting those values using arguments, and then updating only those arguments that need to change. In Section 1.4, for example, we saw that to pass a list to a \texttt{line~} object consisting of two elements (a value to reach and a fixed time in milliseconds in which to reach this value), we could use the \texttt{append} object to append a constant time in milliseconds (see Section 1.4, figure 1.52, file 01_13_conv_interp.maxpat).\textsuperscript{10} Using the \texttt{pack} object, we can obtain the same result, as seen in figure IA.50.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig IA.50.png}
\caption{Two ways to create lists of elements for the \texttt{line~} object}
\end{figure}

In both cases illustrated in the figure, we send the \texttt{line~} object a list of two elements, 0.5 and 20, which “instruct” the \texttt{line~} to reach the number 0.5 in 20 milliseconds.

\textsuperscript{10} If you don’t remember any of this, this is a good opportunity to turn back to Section 1.4 to the section entitled “Conversion of decibel\textsuperscript{s} to Amplitude” to review to discussion, since it used one of the fundamental techniques for interpolating numerical values, and consequently for smoothing and transforming MSP signals.
The objects in the “zl” group can also be created by using the syntax “zl.function_name”. For example, [zl len] using this convention becomes zl.len, [zl slice] becomes zl.slice, and so forth.
Page 163, the beginning of the subsection “THE MULTISLIDER OBJECT” is changed as follows:

THE MULTISLIDER OBJECT

Let’s move now to a graphic object that generates lists of numbers: multislider (seen in figure IA.56). This object can be found in the “Sliders” category of the Object Explorer; click on the “UI Objects” button at the top of the Object Explorer in order to more easily locate its “Sliders” category.

Fig. IA.56 The multislider in the Object Explorer
It is possible to open and remember an audio file in any `sfplay~` object by opening its Inspector and then clicking on the “Choose” button that appears to the right of the attribute; once you’ve selected a file, you can “freeze” your selection by clicking on the “Freeze Attribute” icon, shown graphically as a snowflake, at the bottom of the inspector window. (This group of icons is called the Inspector Toolbar, and can be seen in fig. 3.25.).

After you’ve frozen an attribute, you can be sure that the value will be remembered whenever the patch is loaded. (However, it should be obvious that you can also change the value at any time, either through the Inspector or with an external command.) Many, but not all, attributes can be frozen in this way in the Inspector. If you later want to “unfreeze” the attribute (to forget the saved value the next time the patch is loaded) you can click on the “Unfreeze Attribute” icon, which can be found directly to the right of the “Freeze Attribute” icon.
Page 451, the following footnote is added:

We will conclude the section by introducing two other objects for converting one range into another in the same way as the scale object does, but that operate on signals rather than messages: \texttt{vs.scale\textasciitilde} and \texttt{vs.kscale\textasciitilde}.\footnote{Starting with version 6 of Max, a standard \texttt{scale\textasciitilde} object was introduced which performs the same functions as \texttt{vs.scale\textasciitilde} and \texttt{vs.kscale\textasciitilde}, and which also implements exponential rescaling. We have preserved all references to the objects in the Virtual Sound Macros library in this discussion in order to maintain compatibility with preceding versions of Max.}
Alessandro Cipriani • Maurizio Giri

Electronic Music and Sound Design
Theory and Practice with Max and MSP • volume 1

Topics

“This book is one of the first courses on electronic sound that explicitly integrates perception, theory, and practice using examples of real-time sound synthesis you can manipulate and experience for yourself. In my opinion, Cipriani and Giri have done a masterful job of allowing experiential and theoretical knowledge to reinforce each other. This book will work either as a textbook or as a vehicle for the independent learner. As a bonus, the book includes a thorough introduction to digital signal processing with Max and serves as a wonderful introduction to the programming concepts in that software. I hope you will take advantage of the excellent Max examples the authors have created. They are simultaneously fun and enlightening, and they sound good enough to use on stage. They are also worth examining as models for your own Max patches, or for extending in new ways. As you will see, the theoretical chapters are the “T” chapters, while practical and experiential knowledge is imparted by the “P” chapters. These chapters alternate, in the form of a ladder, refining the concepts at ever higher levels of sophistication. I want to wish you good luck on this new adventure, and also thank my two Italian friends for creating such a comprehensive resource for learning about digital music – the one I wish existed when I was a student!”
(from the Foreword by David Zicarelli, publisher of Max).

This is the first of a series of three volumes dedicated to digital synthesis and sound design. It is ideal for those who are newcomers to the field, but it will also prove to be an indispensable resource for those who wish to deepen existing skills in sound design, in electronic music, or in Max.

ALESSANDRO CIPRIANI — co-authored “Virtual Sound”, a textbook on Csound programming, and was a co-creator of the first online course on sound synthesis available in Europe. His electroacoustic and multimedia compositions have been performed at major festivals and electronic music venues (such as Synthèse Bourges, Venice Biennale and the International Computer Music Conference), and released on CDs and DVDs issued by Computer Music Journal, ICMC, CNI, Everglade, as well as others. He is a tenured professor in electronic music at the Conservatory of Frosinone, a founding member of Edison Studio in Rome, and a member of the editorial board of the journal Organised Sound (published by Cambridge University Press). He has given seminars at many European and American universities, including the University of California - Santa Barbara, Stabiulc Academy in Helsinki, and Accademia di S. Cecilia in Rome.

MAURIZIO GIRI is a professor of composition as well as a teacher of Max/MSP programming techniques at the conservatories of Rome and Frosinone. He is an instrumental, electroacoustic and multimedia composer of music, specializing in digital sound and video processing and computer-assisted composition. He has written computer applications for algorithmic composition and live performance, and has published numerous tutorials on MaxMSP. He was artist in residence at the Cité Internationale des Arts in Paris, and at GRAME in Lyon. He is also an associate member of the Nicod Institute at the Ècole Normale Supérieur in Paris, where he is currently collaborating with other faculty members on a project on the philosophy of sound, entitled “Knowledge of Sound”.

www.contempomem.com

Alessandro Cipriani • Maurizio Giri

Electronic Music and Sound Design
Theory and Practice with Max and MSP • volume 1

UPDATED FOR Max 6

www.contempomem.com