MAX

Reference

Copyright and Trademark Notices

Credits

Original Max Documentation: Chris Dobrian

Max 4.3 Reference Manual: David Zicarelli, Gregory Taylor, Joshua Kit Clayton, jhno, Richard Dudas

Max 4.3 Tutorials and Topics Manual: David Zicarelli, Gregory Taylor, Jeremy Bernstein, Adam Schabtach, Richard Dudas

Max 4.3 Manual page example patches: R. Luke DuBois, Darwin Grosse, Ben Nevile, Joshua Kit Clayton, David Zicarelli

Cover Design: Lilli Wessling Hart

Graphic Design: Gregory Taylor

Introduction

This volume, Max Reference, contains information about each individual Max object. It includes:

Max Objects

Contains precise technical information on the workings of each of the built-in and external objects supplied with Max, organized in alphabetical order.

Max Object Thesaurus

Consists of a reverse index of Max objects, alphabetized by keyword rather than by object name. Use this Thesaurus when you want to know what object(s) are appropriate for the task you are trying to accomplish, then look up those objects by name in the *Objects* section.

Manual Conventions

The central building block of Max is the object. Names of objects are always displayed in bold type, **like this**.

Messages (the arguments that are passed to and from objects) are displayed in plain type, like this.

The name of a Max object displayed in blue type **like this** is hyperlinked to the reference page for that object in this document. Clicking on the blue text will jump to the reference page for that object.

In the "See Also" sections, anything in regular type is a reference to a section of either this manual or the Max Tutorials and Topics manual.

Reading the manual online

The table of contents of the Max Reference Manual is bookmarked, so you can view the bookmarks and jump to any topic listed by clicking on its names. To view the bookmarks, choose Bookmarks from the Windows menu. Click on the triangle next to each section to expand it.

Instead of using the Index at the end of the manual, it might be easier to use Acrobat Reader's Find command. Choose Find from the Tools menu, then type in a word you're looking for. Find will highlight the first instance of the word, and Find Again takes you to subsequent instances. We'd like to take this opportunity to discourage you from printing out the manual unless you find it absolutely necessary.

The **!-** object functions just like the **-** object, but the inlets' functions are reversed.

Input

int	In left inlet: The number is stored, and will be subtracted from a number received in the right inlet. In right inlet: The number in the left inlet is subtracted from the number, and the result is sent out the outlet.
float	Converted to int, unless **!-** has a float argument.
bang	In left inlet: Performs the subtraction with the numbers currently stored. If there is no argument, **!-** initially holds 0.

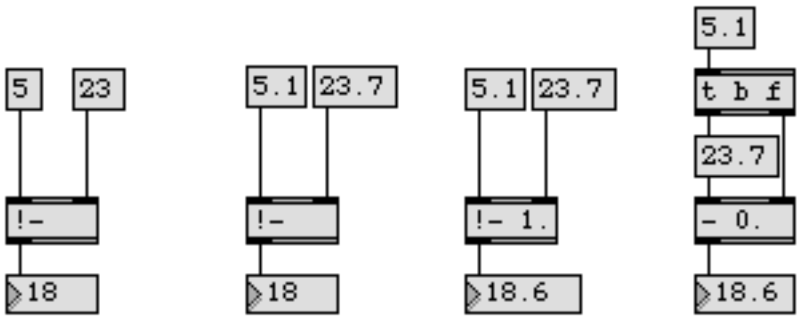
Arguments

int or float	Optional. Sets the initial value, to be subtracted from a number received in the left inlet. Float argument causes the numbers to be subtracted as floats.

Output

int	The difference between the two numbers received in the inlets.
float	Only if there is an argument with a decimal point.

Examples



- *with the inputs swapped*

See Also

expr	Evaluate a mathematical expression
!/	Division object (inlets reversed)
!=	Compare two numbers, output 1 if they are not equal
+	Add two numbers, output the result
-	Subtract two numbers, output the result
** * **	Multiply two numbers, output the result
/	Divide two numbers, output the result
%	Divide two numbers, output the remainder
Tutorial 8	Doing math in Max

!/

The **!/** object functions just like the **/** object, but the inlets' functions are reversed.

Input

int	In left inlet: The number is stored as the *divisor* (the number to be divided into the number in the right inlet). In right inlet: The number is divided by the number in the right inlet, and the result is sent out the outlet.
float	Converted to int, unless **!/** has a float argument.
bang	In left inlet: Performs the division with the numbers currently stored.

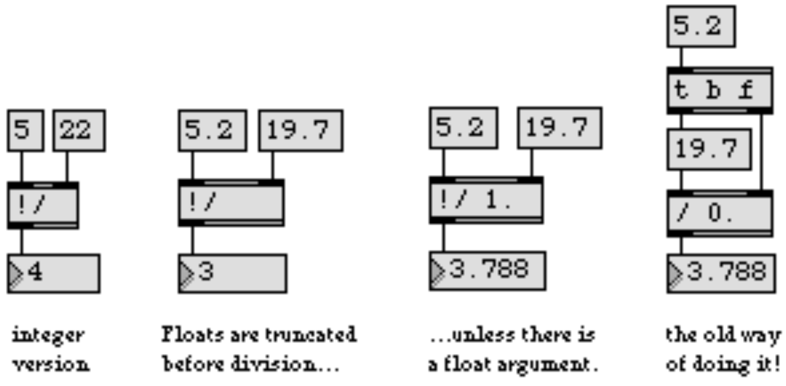
Arguments

int or float	Optional. Sets an initial value for the divisor. If there is no argument, the divisor is set to 1 initially. Float argument causes the numbers to be divided as floats. (Division by 0 is not allowed. Int division by 0 will have the same result as dividing by 1. Float division by 0 will always cause an output of -2^{31}.)

Output

int	The two numbers in the inlets are divided, and the result is sent out the outlet.
float	Only if there is an argument with a decimal point.

Examples



***/** with the inputs swapped*

See Also

expr	Evaluate a mathematical expression
!-	Subtraction object (inlets reversed)
!=	Compare two numbers, output 1 if they are not equal
+	Add two numbers, output the result
-	Subtract two numbers, output the result
*****	Multiply two numbers, output the result
/	Divide two numbers, output the result
%	Divide two numbers, output the remainder
Tutorial 8	Doing math in Max

!=

Input

int	In left inlet: The number is compared with the number in the right inlet. If the two numbers are not equal, **!=** outputs 1. If they are equal **!=** outputs 0. In right inlet: The number is stored, to be compared with a number received in the left inlet.
float	Converted to int before comparison, unless **!=** has a float argument.
bang	In left inlet: Performs the comparison with the numbers currently stored. If there is no argument, **!=** initially holds 0 for comparison.
list	In left inlet: Compares first and second number, outputs 1 if they are not equal, 0 if they are equal.

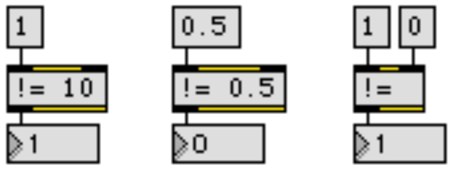
Arguments

int or float	Optional. Sets the initial value, to be compared with a number received in the left inlet. Float argument forces a float comparison.

Output

int	1 if the numbers in the inlets are not equal, 0 if they are equal.

Examples



Test if two numbers are not *equal*

See Also

select	Select certain inputs, pass the rest on
split	Look for a range of numbers
<	*Is less than*, comparison of two numbers
<=	*Is less than or equal to*, comparison of two numbers
==	Compare two numbers, output 1 if they are equal
>	*Is greater than*, comparison of two numbers
>=	*Is greater than or equal to*, comparison of two numbers
Tutorial 15	Making decisions with comparisons

+

Add two numbers, output the result

Input

int	In left inlet: The number is added to the number in the right inlet, and the result is sent out the outlet. In right inlet: The number is stored for addition to a number received in the left inlet.
float	Converted to int, unless **+** has a float argument.
bang	In left inlet: Performs the addition with the numbers currently stored. If there is no argument, **+** initially holds 0.
list	In left inlet: The first number is added to the second number, and the result is sent out the outlet.
set	In left inlet: The word set, followed by a number, adds that number to the number in the right inlet but nothing is sent out. A subsequent bang sends out the result.

The set message functions similarly for all the arithmetic operators, logical operators, and bitwise operators: **+**, **-**, *****, **/**, **%**, **<**, **<=**, **==**, **!-**, **!/**, **!=**, **>=**, **>**, **&&**, **||**, **&**, **|**, **<<**, and **>>**. The number is used as the left operand, and the expression is evaluated, but the result is not sent out.

Arguments

int or float	Optional. Sets the initial value, to be added to a number received in the left inlet. Float argument causes the numbers to be added as floats.

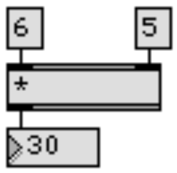
Output

int	The sum of the two numbers received in the inlets.
float	Only if there is an argument with a decimal point.

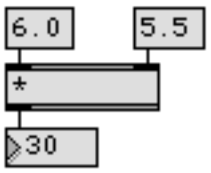
+

Add two numbers, output the result

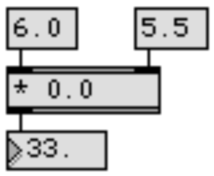
Examples



Normally adds ints



Floats are truncated before addition...



unless there is a float argument

See Also

expr	Evaluate a mathematical expression
!-	Subtraction object (inlets reversed)
!/	Division object (inlets reversed)
-	Subtract two numbers, output the result
*****	Multiply two numbers, output the result
/	Divide two numbers, output the result
%	Divide two numbers, output the remainder
Tutorial 8	Doing math in Max

-

Input

int	In left inlet: The number in the right inlet is subtracted from the number, and the result is sent out the outlet. In right inlet: The number is stored, to be subtracted from a number received in the left inlet.
float	Converted to int, unless **-** has a float argument.
bang	In left inlet: Performs the subtraction with the numbers currently stored. If there is no argument, **-** initially holds 0.
list	In left inlet: The second number is subtracted from the first number, and the result is sent out the outlet.

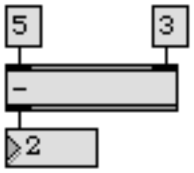
Arguments

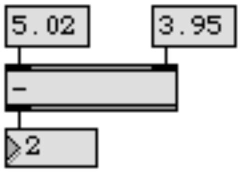
int or float	Optional. Sets the initial value, to be subtracted from a number received in the left inlet. Float argument causes the numbers to be subtracted as floats.

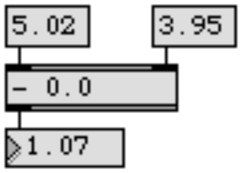
Output

int	The difference between the two numbers received in the inlets.
float	Only if there is an argument with a decimal point.

Examples







Subtracted as ints *Floats are truncated before subtraction…* *…unless there is a float argument*

See Also

expr	Evaluate a mathematical expression
!-	Subtraction object (inlets reversed)
!/	Division object (inlets reversed)
+	Add two numbers, output the result
-	Subtract two numbers, output the result
*****	Multiply two numbers, output the result
/	Divide two numbers, output the result
%	Divide two numbers, output the remainder
Tutorial 8	Doing math in Max

*

Multiply two numbers, output the result

Input

int	In left inlet: The number is multiplied by the number in the right inlet, and the result is sent out the outlet.
	In right inlet: The number is stored for multiplication with a number received in the left inlet.
float	Converted to int before multiplication, unless * has a float argument.
bang	In left inlet: Performs the multiplication with the numbers currently stored. If there is no argument, * initially holds 0 as a multiplier.
list	In left inlet: The first number is multiplied by the second number, and the result is sent out the outlet.

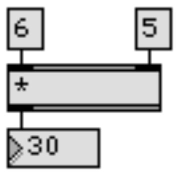
Arguments

int or float	Optional. Sets the initial value, to be multiplied by a number received in the left inlet. Float argument causes the numbers to be multiplied as floats.

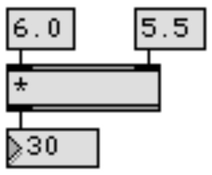
Output

int	The product of the two numbers received in the inlets.
float	Only if there is an argument with a decimal point.

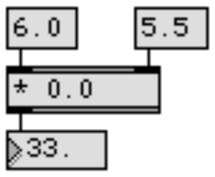
Examples



Multiplied as ints



Floats are truncated before multiplication…



…unless there is a float argument

See Also

expr	Evaluate a mathematical expression
!-	Subtraction object (inlets reversed)
!/	Division object (inlets reversed)
!=	Compare two numbers, output 1 if they are not equal
+	Add two numbers, output the result
-	Subtract two numbers, output the result
/	Divide two numbers, output the result
%	Divide two numbers, output the remainder
<<	Shift all bits to the left
Tutorial 8	Doing math in Max

/

Divide two numbers, output the result

Input

int	In left inlet: The number is divided by the number in the right inlet, and the result is sent out the outlet. In right inlet: The number is stored as the *divisor* (the number to be divided into the number in the left inlet).
float	Converted to int, unless **/** has a float argument.
bang	In left inlet: Performs the division with the numbers currently stored.
list	In left inlet: The first number is divided by the second number, and the result is sent out the outlet.

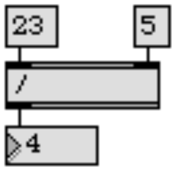
Arguments

int or float	Optional. Sets an initial value for the divisor. If there is no argument, the divisor is set to 1 initially. Float argument causes the numbers to be divided as floats. (Division by 0 is not allowed. Int division by 0 will have the same result as dividing by 1. Float division by 0 will always cause an output of -2^{31}.)

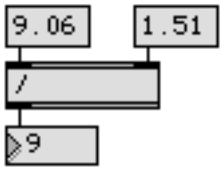
Output

int	The two numbers in the inlets are divided, and the result is sent out the outlet.
float	Only if there is an argument with a decimal point.

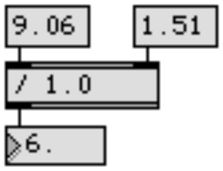
Examples



Remainder is discarded



Floats are truncated before division…



…unless there is a float argument

See Also

expr	Evaluate a mathematical expression
!-	Subtraction object (inlets reversed)
!/	Division object (inlets reversed)
+	Add two numbers, output the result
-	Subtract two numbers, output the result
*****	Multiply two numbers, output the result
%	Divide two numbers, output the remainder
Tutorial 8	Doing math in Max

%

Divide two numbers, output remainder

Input

int	In left inlet: The number is divided by the number in the right inlet, and the *remainder* is sent out the outlet. In right inlet: The number is stored as the *divisor* (the number to be divided into the number in the left inlet) for calculating the remainder.
float	Converted to int.
bang	In left inlet: Performs the operation with the numbers currently stored.
list	In left inlet: The first number is divided by the second number, and the remainder is sent out the outlet.

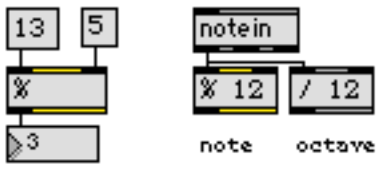
Arguments

int	Optional. Sets an initial value for the divisor. If there is no argument, the divisor is set to 1 initially.

Output

int	When the two numbers in the inlets are divided, the remainder is sent out the outlet. **%** is called the *modulo* operator.

Examples



Find the remainder of a division

See Also

expr	Evaluate a mathematical expression
!-	Subtraction object (inlets reversed)
!/	Division object (inlets reversed)
+	Add two numbers, output the result
-	Subtract two numbers, output the result
*****	Multiply two numbers, output the result
/	Divide two numbers, output the result
Tutorial 8	Doing math in Max

Input

int	In left inlet: If the number is less than the number in the right inlet, **<** outputs 1. Otherwise, **<** outputs 0.
	In right inlet: The number is stored to be compared with a number received in the left inlet.
float	Converted to int before comparison, unless **<** has a float argument.
bang	In left inlet: Performs the comparison with the numbers currently stored. If there is no argument, **<** initially holds 0 for comparison.
list	In left inlet: If the first number *is less than* the second number, **<** outputs 1. Otherwise, **<** outputs 0.

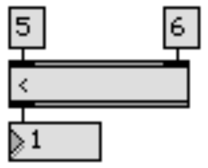
Arguments

int or float	Optional. Sets the initial value, to be compared with a number received in the left inlet. Float argument forces a float comparison.

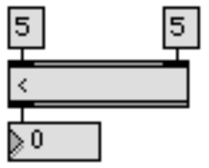
Output

int	1 if the number in the left inlet is less than the number in the right inlet. 0 if the number in the left inlet is *greater than or equal to* the number in the right inlet.

Examples



Number on left is less than number on right



Number on left is not less than number on right

See Also

!=	Compare two numbers, output 1 if they are not equal
<=	*Is less than or equal to,* comparison of two numbers
==	Compare two numbers, output 1 if they are equal
>	*Is greater than,* comparison of two numbers
>=	*Is greater than or equal to,* comparison of two numbers
Tutorial 15	Making decisions with comparisons

<=

Is less than or equal to, comparison of two numbers

Input

int	In left inlet: If the number *is less than or equal to* the number in the right inlet, **<=** outputs 1. Otherwise, **<=** outputs 0.
	In right inlet: The number is stored to be compared with a number received in the left inlet.
float	Converted to int before comparison, unless **<=** has a float argument.
bang	In left inlet: Performs the comparison with the numbers currently stored. If there is no argument, **<=** initially holds 0 for comparison.
list	In left inlet: If the first number *is less than or equal to* the second number, **<=** outputs 1. Otherwise, **<=** outputs 0.

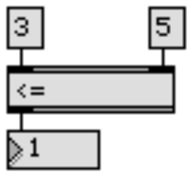
Arguments

int or float	Optional. Sets the initial value, to be compared with a number received in the left inlet. Float argument forces a float comparison.

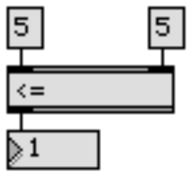
Output

int	1 if the number in the left inlet is less than or equal to the number in the right inlet. 0 if the number in the left inlet *is greater than* the number in the right inlet.

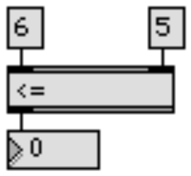
Examples



Is less than...



or equal to



Is not less than or equal to

See Also

!=	Compare two numbers, output 1 if they are not equal
<	*Is less than*, comparison of two numbers
==	Compare two numbers, output 1 if they are equal
>	*Is greater than*, comparison of two numbers
>=	*Is greater than or equal to*, comparison of two numbers
Tutorial 15	Making decisions with comparisons

==

Compare two numbers, output 1 if they are equal

Input

int	In left inlet: The number is compared with the number in the right inlet. If the two numbers are equal, **==** outputs 1. If they are not equal **==** outputs 0. In right inlet: The number is stored to be compared with a number received in the left inlet.
float	Converted to int before comparison, unless **==** has a float argument.
bang	In left inlet: Performs the comparison with the numbers currently stored. If there is no argument, **==** initially holds 0 for comparison.
list	In left inlet: Compares first and second number, outputs 1 if they are equal, 0 if they are not equal.

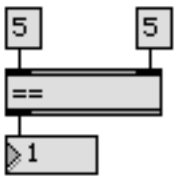
Arguments

int or float	Optional. Sets the initial value, to be compared with a number received in the left inlet. Float argument forces a float comparison.

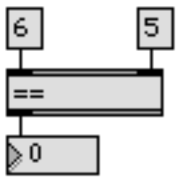
Output

int	1 if the numbers in the inlets are equal, 0 if they are not equal.

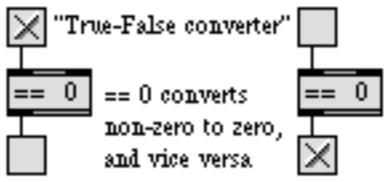
Examples



The numbers are equal



The numbers are not equal



Using **==** 0 *as a logical "not"*

See Also

select	Select certain inputs, pass the rest on
split	Look for a range of numbers
!=	Compare two numbers, output 1 if they are not equal
<	*Is less than*, comparison of two numbers
<=	*Is less than or equal to*, comparison of two numbers
>	*Is greater than*, comparison of two numbers
>=	*Is greater than or equal to*, comparison of two numbers
Tutorial 15	Making decisions with comparisons

Input

int	In left inlet: If the number *is greater than* the number in the right inlet, **>** outputs 1. Otherwise, **>** outputs 0.
	In right inlet: The number is stored to be compared with a number received in the left inlet.
float	Converted to int before comparison, unless **>** has a float argument.
bang	In left inlet: Performs the comparison with the numbers currently stored. If there is no argument, **>** initially holds 0 for comparison.
list	In left inlet: If the first number *is greater than* the second number, **>** outputs 1. Otherwise, **>** outputs 0.

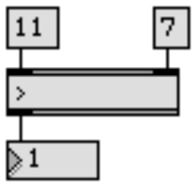
Arguments

int or float	Optional. Sets the initial value, to be compared with a number received in the left inlet. Float argument forces a float comparison.

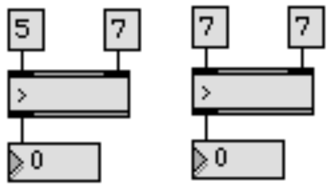
Output

int	1 if the number in the left inlet is greater than the number in the right inlet. 0 if the number in the left inlet *is less than or equal to* the number in the right inlet.

Examples



The number on the left is greater



The number on the left is not greater

See Also

!=	Compare two numbers, output 1 if they are not equal
<	*Is less than*, comparison of two numbers
<=	*Is less than or equal to*, comparison of two numbers
==	Compare two numbers, output 1 if they are equal
>=	*Is greater than or equal to*, comparison of two numbers
Tutorial 15	Making decisions with comparisons

>=

Is greater than or equal to, comparison of two numbers

Input

int	In left inlet: If the number is *greater than or equal to* the number in the right inlet, **>=** outputs 1. Otherwise, **>=** outputs 0. In right inlet: The number is stored to be compared with a number received in the left inlet.
float	Converted to int before comparison, unless **>=** has a float argument.
bang	In left inlet: Performs the comparison with the numbers currently stored. If there is no argument, **>=** initially holds 0 for comparison.
list	In left inlet: If the first number is *greater than or equal to* the second number, **>=** outputs 1. Otherwise, **>=** outputs 0.

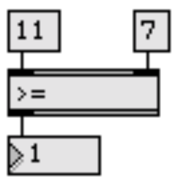
Arguments

int or float	Optional. Sets the initial value, to be compared with a number received in the left inlet. Float argument forces a float comparison.

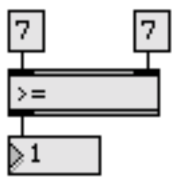
Output

int	1 if the number in the left inlet *is greater than or equal to* the number in the right inlet. 0 if the number in the left inlet *is less than* the number in the right inlet.

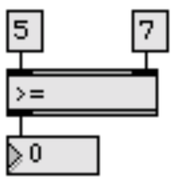
Examples



Is greater than...



or equal to



Is not greater than or equal to

See Also

!=	Compare two numbers, output 1 if they are not equal
<	*Is less than*, comparison of two numbers
<=	*Is less than or equal to*, comparison of two numbers
==	Compare two numbers, output 1 if they are equal
>	*Is greater than*, comparison of two numbers
Tutorial 15	Making decisions with comparisons

&

Input

int	In left inlet: The number is compared, in binary form, with the number in the right inlet. The output is a number composed of those bits which are 1 in *both* numbers.
	In right inlet: The number is stored for comparison with a number received in the left inlet.
float	Converted to int.
bang	In left inlet: Performs the comparison with the numbers currently stored. If there is no argument, **&** initially holds 0 for comparison.
list	In left inlet: Compares the first and second numbers bit-by-bit, and outputs a number composed of those bits which are 1 in both numbers.

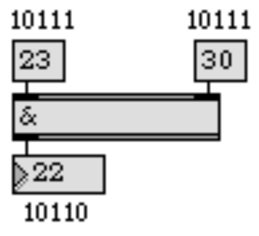
Arguments

int	Optional. Sets an initial value to be compared with a number received in the left inlet.

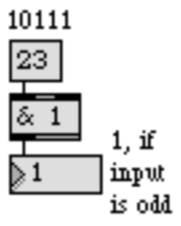
Output

int	The two numbers received in the inlets are compared, one bit at a time. If a bit is 1 in both numbers, it will be 1 in the output number, otherwise it will be 0 in the output number.

Examples



Nonzero bits shared by both numbers



Can be used as an odd/even detector

See Also

&&	If both numbers are non-zero, output 1
|	Bitwise union of two numbers
||	If either of two numbers is non-zero, output 1

&&

If both numbers are non-zero, output 1

Input

int	If the number in *both* inlets is not 0, then the output is 1. If the number in one or both of the inlets is 0, then the output is 0. A number in the left inlet triggers the output.
float	Converted to int.
bang	In left inlet: Performs the operation with the numbers currently stored. If there is no argument, **&&** initially holds 0.
list	In left inlet: If both the first and second numbers are not 0, then the output is 1. Otherwise, the output is 0.

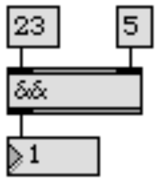
Arguments

int	Optional. Sets an initial value to be stored by **&&**. A number in the right inlet changes the value set by the argument.

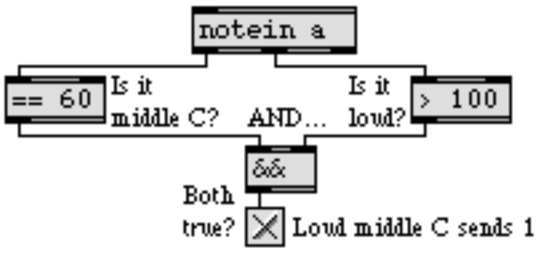
Output

int	If the number in the left inlet *and* the number in the right inlet (or specified by the argument) are both not 0, then the output is 1. Otherwise, the output is 0.

Examples



Both numbers are not 0



Used to combine comparisons

See Also

&	Bitwise intersection of two numbers
|	Bitwise union of two numbers
||	If either of two numbers is non-zero, output 1
Tutorial 15	Making decisions with comparisons

Input

int	In left inlet: Outputs a number composed of all those bits which are 1 in either of the two numbers. In right inlet: The number is stored for combination with a number received in the left inlet.
float	Converted to int.
bang	In left inlet: Performs the calculation with the numbers currently stored. If there is no argument, **|** initially holds 0.
list	In left inlet: Combines the first and second numbers bit-by-bit, and outputs a number composed of all those bits which are 1 in either of the two numbers.

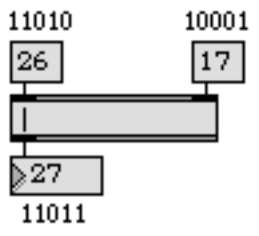
Arguments

int	Optional. Sets an initial value to be or-ed with a number received in the left inlet.

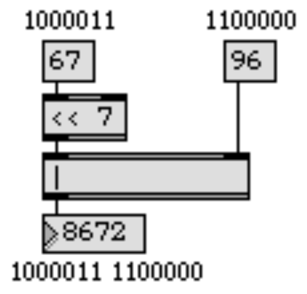
Output

int	All the nonzero bits of the two numbers received in the inlets are combined. If a bit is 1 in either one of the numbers, it will be 1 in the output number, otherwise it will be 0 in the output number.

Examples



All non-zero bits are combined



Can be used to pack two numbers into one int

See Also

&	Bitwise intersection of two numbers
&&	If both numbers are non-zero, output 1
||	If either of two numbers is non-zero, output 1

Input

int	If the number in *either* inlet is not 0, then the output is 1. If the number in *both* of the inlets is 0, then the output is 0. A number in the left inlet triggers the output.
float	Converted to int.
bang	In left inlet: Performs the operation with the numbers currently stored. If there is no argument, **||** initially holds 0.
list	In left inlet: If either the first or second number is not 0, then the output is 1. Otherwise, the output is 0.

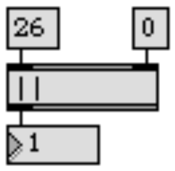
Arguments

int	Optional. Sets an initial value to be stored by **||**. A number in the right inlet changes the value set by the argument.

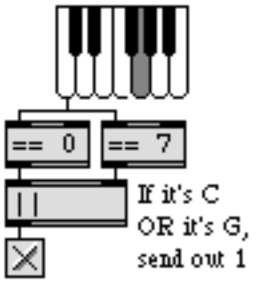
Output

int	If either the number in the left inlet *or* the number in the right inlet (or specified by the argument) is not 0, then the output is 1. Otherwise, the output is 0.

Examples



One of the numbers is not 0



Used to combine comparisons

See Also

&	Bitwise intersection of two numbers
&&	If both numbers are non-zero, output 1
|	Bitwise union of two numbers
Tutorial 15	Making decisions with comparisons

<<

Input

int	In left inlet: All bits of the number, in binary form, are shifted to the left by a certain number of bits. The resulting number is sent out the outlet. In right inlet: The number is stored as the number of bits to left-shift the number in the left inlet.
float	Converted to int.
bang	In left inlet: Performs the bit-shift with the numbers currently stored. If there is no argument, **<<** initially holds 0 as the number of bits by which to shift.
list	In left inlet: The first number is bit-shifted to the left by the number of bits specified by the second number.

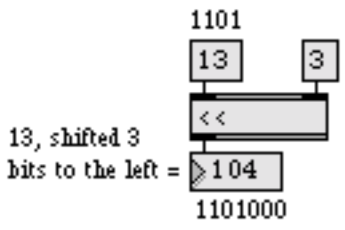
Arguments

int	Optional. Sets an initial value for the number of bits by which to shift leftward.

Output

int	The number in the left inlet is bit-shifted to the left by a certain number of bits. The number of bits by which to shift is specified by the number in the right inlet. The output is the resulting bit-shifted number.

Examples



Same effect as multiplying by a power of 2

See Also

*	Multiply two numbers, output the result
>>	Shift all bits to the right

>>

Input

int	In left inlet: All bits of the number, in binary form, are shifted to the right by a certain number of bits. The resulting number is sent out the outlet. In right inlet: The number is stored as the number of bits to right-shift the number in the left inlet.
float	Converted to int.
bang	In left inlet: Performs the bit-shift with the numbers currently stored. If there is no argument, **>>** initially holds 0 as the number of bits by which to shift.
list	In left inlet: The first number is bit-shifted to the right by the number of bits specified by the second number.

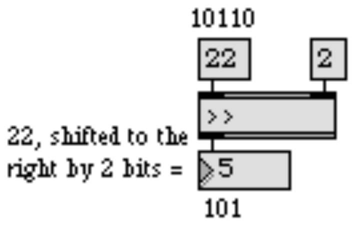
Arguments

int	Optional. Sets an initial value for the number of bits by which to shift rightward.

Output

int	The number in the left inlet is bit-shifted to the right by a certain number of bits. The number of bits by which to shift is specified by the number in the right inlet. The output is the resulting bit-shifted number.

Examples



Same effect as dividing by a power of 2

See Also

!/	Division object (inlets reversed)
<<	Shift all bits to the left

abs

Output the absolute value of the input

Input

int	The absolute (non-negative) value of the input is sent out the output.
float	Converted to int, unless **abs** has a float argument.
int or float	Optional. Float argument forces a float output.

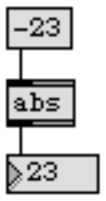
Arguments

int or float	Optional. Float argument forces a float output.

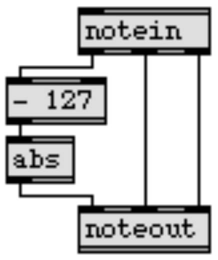
Output

int	The absolute value of the input.
float	Only if there is an argument with a decimal point.

Examples



Output is nonnegative



Used here to invert input

See Also

expr	Evaluate a mathematical expression
Tutorial 14	Sliders and dials

absolutepath

Convert a file name to an absolute path

Input

any symbol	A file name or path as a symbol. Input pathnames can contain slashes, colons, or backslashes as separators. The **absolutepath** object converts a file name or path to an absolute path, resolving any aliases in doing so.

Arguments

None.

Output

any symbol	If the incoming file name or path is found, the output is an absolute path. The output pathnames contain slash separators.

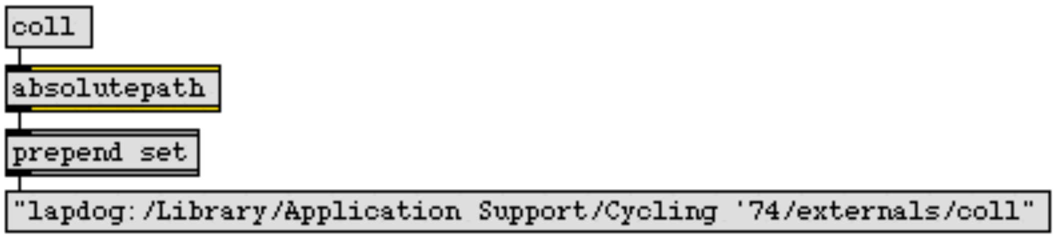
Absolute pathnames look like this:

"C:/Max Folder/extras/mystuff/mypatch.pat"

The **conformpath** object can be used to convert paths of one pathtype and/or pathstyle to another.

If the file is not found, **absolutepath** outputs the symbol notfound.

Examples



```
coll
absolutepath
prepend set
"lapdog:/Library/Application Support/Cycling '74/externals/coll"
```

See Also

conformpath	Convert paths of one pathtype and/or pathstyle to another
dropfile	Define a region for dragging and dropping a file
opendialog	Open a dialog to ask for a file or folder
relativepath	Convert an absolute to a relative path
savedialog	Open a dialog to ask for a filename for saving
strippath	Get a filename from a full pathname
File Preferences	

acos

Input

float or int	Input to a arc-cosine function.
bang	In left inlet: Calculates the arc-cosine of the number currently stored. If there is no argument, **acos** initially holds 0.

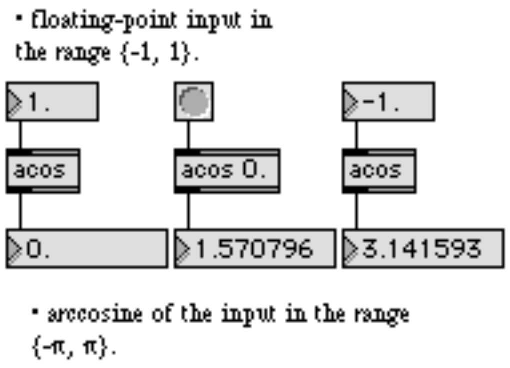
Arguments

float or int	Optional. Sets the initial value for the arc-cosine function.

Output

float or int	The arc-cosine of the input.

Examples


• floating-point input in the range {-1, 1}.
1.
acos
acos 0.
-1.
0.
1.570796
3.141593
• arccosine of the input in the range {-π, π}.

See Also

acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

acosh

Input

float or int	Input to a hyperbolic arc-cosine function.
bang	In left inlet: Calculates a hyperbolic arc-cosine of the number currently stored. If there is no argument, **acosh** initially holds 0.

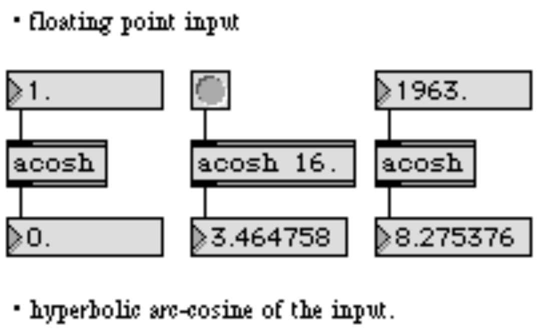
Arguments

float or int	Optional. Sets the initial value for the hyperbolic arc-cosine function.

Output

float or int	The hyperbolic arc-cosine of the input.

Examples



See Also

acos	Arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

accum

Store, add to, and multiply a number

Input

int	In left inlet: Replaces the value stored in **accum**, and sends the new value out the outlet.
	In middle inlet: The number is added to the stored value, without triggering output.
	In right inlet: The stored value is multiplied by the input, without triggering output.
float	In left and middle inlet: Converted to int, unless **accum** has a float argument.
	In right inlet: Multiplication is done with floats, even if the value is stored as an int.
bang	In left inlet: Outputs the value currently stored in **accum**.
set	In left inlet: The word set, followed by a number, sets the stored value to that number, without triggering output.

Arguments

int or float	Optional. Sets the initial value stored in **accum**. An argument with a decimal point causes the value to be stored as a float.

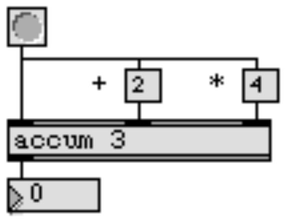
Output

int	The value currently held by **accum**.
float	Only if there is an argument with a decimal point.

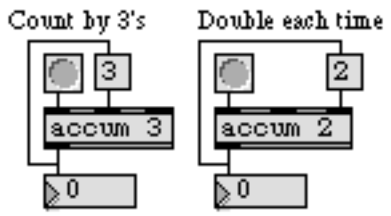
accum

Store, add to, and multiply a number

Examples



Add to and/or multiply a stored value



Used here to increment by different amounts

See Also

counter	Count the bang messages received, output the count
float	Store a decimal number
int	Store an integer value
Tutorial 21	Storing numbers

active

Send 1 when patcher window is active,
0 when inactive

Input

There are no inlets. Output is triggered automatically when the patcher window is activated or deactivated.

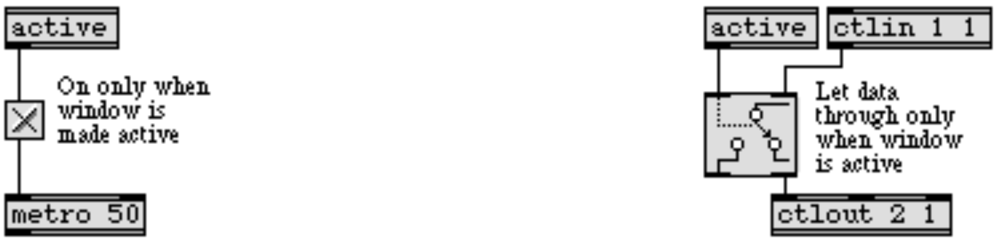
Arguments

None.

Output

int When the patcher window that contains **active** is activated, **active** sends out 1. When the window is made inactive, **active** sends out 0.

Examples


active
On only when window is made active
metro 50
active
ctlin 1 1
Let data through only when window is active
ctlout 2 1

Turn on a process or open a gate when the window is made active

See Also

closebang	Send a bang when patcher window is closed
loadbang	Send a bang automatically when patch is loaded
Tutorial 40	Automatic actions

anal

Make a histogram of number pairs received

Input

int	Reports how many times this number and the previously received number have occurred in immediate succession. (The first time a number is received, there has been no previous number, so nothing happens.)
reset	Erases the most recently received number from the memory of the **anal** object. The next number to be received gets stored in its place, to serve as the next "previous" value (but nothing else happens).
clear	Erases the memory of the **anal** object entirely, but retains the most recently received number to use as the next "previous" value.

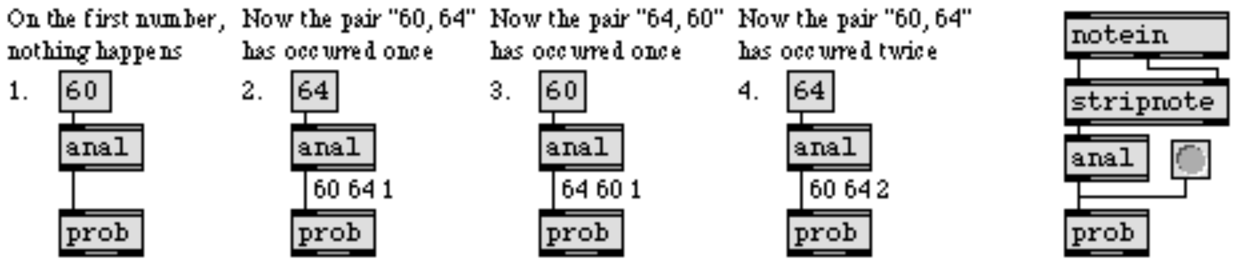
Arguments

int	Optional. Sets a maximum limit for how many different number pairs can be kept track of by **anal**. The maximum number of different pairs is 1024. If no argument is present, **anal** can store up to 128 different pairs.

Output

list	The first two numbers in the list are the two most recently received numbers, and the third number shows how many times that particular succession of two numbers has been received. This list of three numbers is designed to be used as input to the **prob** object, to create a probability matrix of transitions from one number to another (known as a first-order Markov chain).

Examples



On the first number, nothing happens
1.
60
anal
prob
Now the pair "60, 64" has occurred once
2.
64
anal
60 64 1
prob
Now the pair "64, 60" has occurred once
3.
60
anal
64 60 1
prob
Now the pair "60, 64" has occurred twice
4.
64
anal
60 64 2
prob
notein
stripnote
anal
prob

Keep track of number pairs and their relative frequency of occurrence; pass the information to **prob** *to generate similar transitions*

See Also

histo	Make a histogram of the numbers received
prob	Make weighted random series of numbers

append

Append arguments at the end of a message

Input

set	The word set, followed by any message, will replace the message stored in **append**, without triggering output.
anything else	The message stored in **append** is appended, preceded by a space, to the end of any message that is received in the inlet, and the combined message is sent out the outlet.

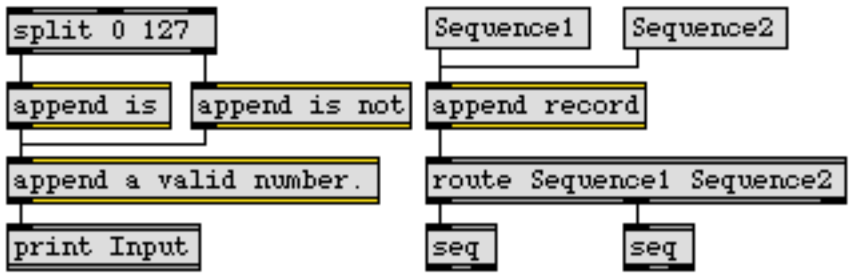
Arguments

anything	Optional. Sets the message that will be appended to the end of incoming messages.

Output

anything	The message received in the inlet is combined with the message stored in **append**, and then sent out the outlet.

Examples



Symbols can be combined into meaningful messages with **append**

See Also

prepend	Put one message at the beginning of another
Tutorial 25	Managing messages

asin

Input

float or int	Input to a arc-sine function.
bang	In left inlet: Calculates the arc-sine of the number currently stored. If there is no argument, **asin** initially holds 0.

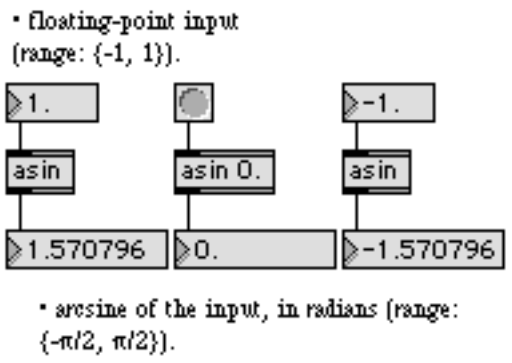
Arguments

float or int	Optional. Sets the initial value for the arc-sine function.

Output

float or int	The arc-sine of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

asinh

Input

float or int	Input to a hyperbolic arc-sine function.
bang	In left inlet: Calculates the hyperbolic arc-sine of the number currently stored. If there is no argument, **asin** initially holds 0.

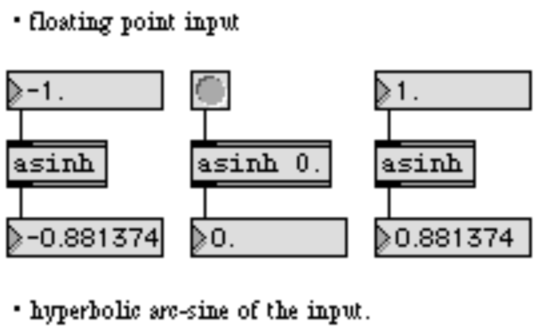
Arguments

float or int	Optional. Sets the initial value for the hyperbolic arc-sine function.

Output

float or int	The hyperbolic arc-sine of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

atan

Input

float or int	Input to a arc-tangent function.
bang	In left inlet: Calculates the arc-tangent of the number currently stored. If there is no argument, **atan** initially holds 0.

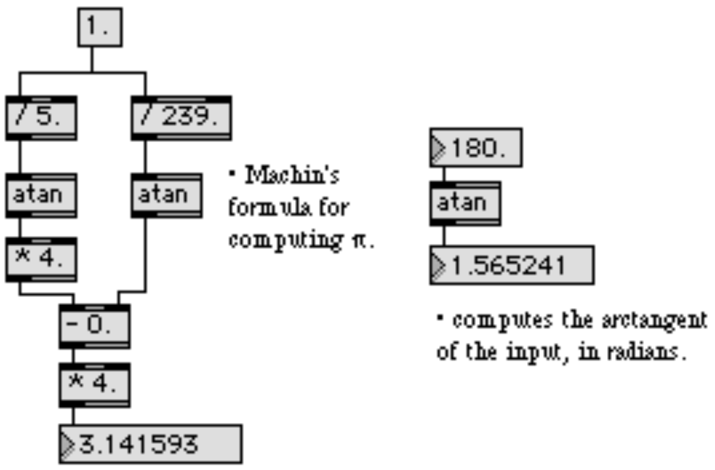
Arguments

float or int	Optional. Sets the initial value for the arc-tangent function.

Output

float or int	The arc-tangent of the input.

Examples


1.
/ 5.
/ 239.
atan
atan
* 4.
- 0.
* 4.
3.141593
• Machin's formula for computing π.
180.
atan
1.565241
• computes the arctangent of the input, in radians.

See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

atan2

Input

float or int — In left input: *x* value input to an arc-tangent function.

In right input: *y* value input to an arc-tangent function.

bang — In left inlet: Calculates the arc-tangent of the numbers currently stored. If there are no arguments, **atan2** initially holds 0 for both input values.

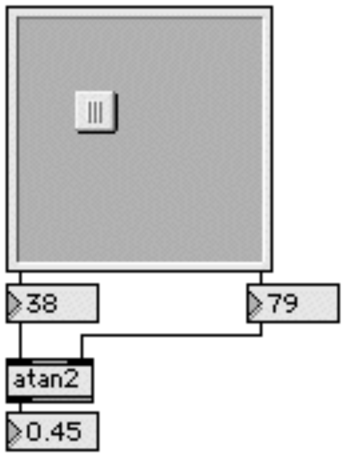
Arguments

float or int — Optional. Two ints may be used to set the initial value for the arc-tangent function.

Output

float or int — The arc-tangent of the input values (i.e. *Arc-tangent(y/x)*).

Examples



• calculates the angle from two points around an origin (atan(y/x)) in radians.

See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

atanh

Input

float or int	Input to a hyperbolic arc-tangent function.
bang	In left inlet: Calculates the hyperbolic arc-tangent of the number currently stored. If there is no argument, **atanh** initially holds 0.

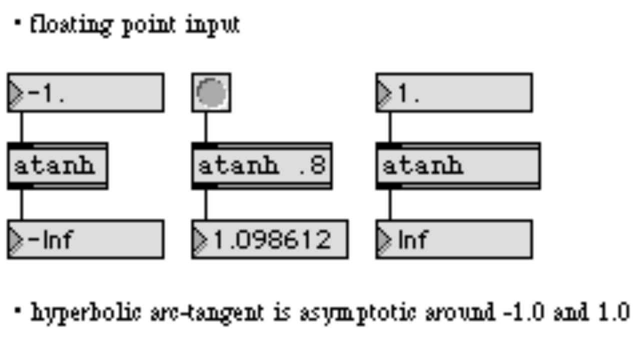
Arguments

float or int	Optional. Sets the initial value for the hyperbolic arc-tangent function.

Output

float or int	The hyperbolic arc-tangent of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

bag

Store a collection of numbers

Input

int	In left inlet: The number is either added to or deleted from the collection of numbers stored in **bag**, depending on the number in the right inlet. In right inlet: The number is stored as an indicator of whether to include or delete the next number received in the left inlet. If non-zero, the number received in the left inlet is added to the **bag**. If 0, the number is deleted from the **bag**. No output is triggered by a number received in either inlet.
float	Converted to int.
bang	In left inlet: Causes **bag** to send all its numbers out the outlet.
clear	In left inlet: Deletes the entire contents of the **bag**.
list	In left inlet: If the second number is not 0, the first number is included in the **bag**. If the second number is 0, the first number is deleted from the **bag**.
send	In left inlet: The word send, followed by the name of a **receive** object, sends the result of a bang message to all **receive** objects with that name, instead of out the **bag** object's outlet.
length	In left inlet: Reports how many numbers are currently stored in the **bag**.
cut	In left inlet: Sends out the oldest (earliest received) number stored in the **bag**, and deletes it from the **bag**.

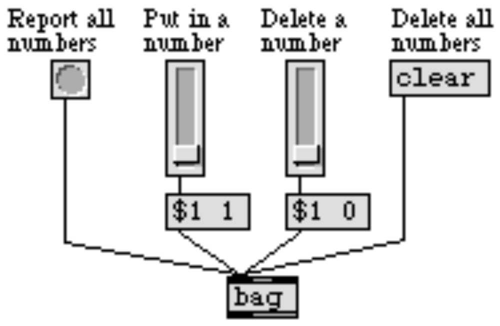
Arguments

any symbol	Optional. Causes **bag** to store duplicate numbers. If there is no argument, **bag** will store only one of each number at a time. The argument must not be a number.

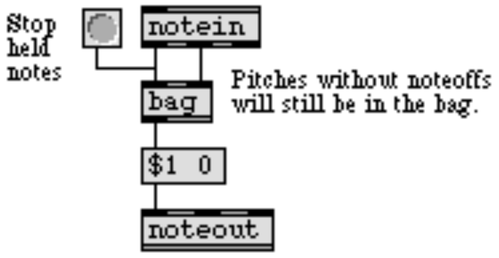
Output

int	When bang is received in the left inlet, all the numbers stored in **bag** are sent out one at a time, in reverse order from that in which they were stored. When cut is received in the left inlet, the oldest stored number is sent out. When length is received in the left inlet, the number of items in the **bag** is sent out.

Examples



Store a collection of numbers



Used here to detect held notes

See Also

coll	Store and edit a collection of different messages
funbuff	Store x,y pairs of numbers
offer	Store x,y pairs of numbers temporarily
Data Structures	Ways of storing data in Max

bangbang / b

Send a bang *to many places, in order*

Input

anything	Causes a bang to be sent out all outlets, in right-to-left order.

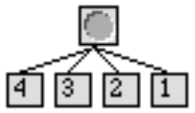
Arguments

int	Optional. Sets the number of outlets. Limited between 1 and 10. Any number greater than 10 is set to 10; any number less than 1 is set to 2. If there is no argument, there will be 2 outlets.
float	Converted to int.

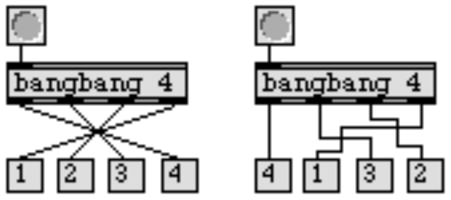
Output

bang	When a message is received in the inlet, bang is sent out each outlet, in order from right to left.

Examples



Order is normally right-to-left



Order is specified by **bangbang**

See Also

button	Flash on any message, send a bang
trigger	Send input to many places, in order
Tutorial 7	Right-to-left order

bendin

Output received MIDI pitch bend values

Input

(MIDI) **bendin** receives its input from a MIDI pitch bend message received from a MIDI input device.

enable The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.

port The word port, followed by a letter a-z or the name of an MIDI port or device, sets the port from which the object receives incoming pitch bend messages. The word port is optional and may be omitted.

(mouse) Double-clicking on a **bendin** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

a-z Optional. Specifies the port from which to receive incoming pitch bend messages. If there is no argument, **bendin** receives from all channels on all ports.

(MIDI name) Optional. The name of a MIDI input device may be used as the first argument to specify the port.

a-z and int A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to receive pitch bend messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.

int A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the MIDI Setup dialog.

Output

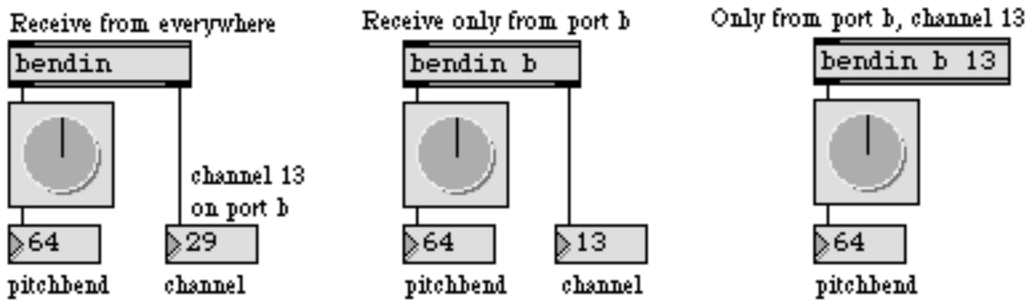
int If a specific channel number is included in the argument, there is only one outlet. The output is the incoming pitch bend value from 0-127 (the most significant byte of the MIDI pitch bend message) on the specified channel and port.

If there is no channel number specified by the argument, **bendin** will have a second outlet, on the right, which will output the channel number of the incoming pitch bend message.

bendin

Output received MIDI pitch bend values

Examples



Pitch bend messages can be received from everywhere, a specific port, or a specific port and channel

See Also

bendout	Transmit MIDI pitch bend messages
ctlin	Output received MIDI control values
midiin	Output received raw MIDI data
notein	Output received MIDI note messages
rtin	Output received MIDI real time messages
xbendout	Prepare extra precision MIDI pitch bend messages
xbendin	Interpret extra precision MIDI pitch bend messages
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified
Tutorial 16	More MIDI ins and outs

bendout

Transmit MIDI pitch bend messages

Input

int	In left inlet: The number is transmitted as a MIDI pitch bend value on the specified channel and port. Numbers are limited between 0 and 127.
	In right inlet: The number is stored as the channel number on which to transmit the pitch bend messages.
float	Converted to int.
list	In left inlet: The first number is the pitch bend value, and the second number is the channel, of a MIDI pitch bend message, transmitted on the specified channel and port.
enable	The message enable 0 disables the object, causing it not to transmit MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	In left inlet: The word port, followed by a letter a-z or the name of a MIDI output port or device, specifies the port used to transmit MIDI messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **bendout** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

a-z	Optional. Specifies the port for transmitting MIDI pitch bend messages. Channel numbers greater than 16 received in the right inlet will be *wrapped around* to stay within the 1-16 range. If there is no argument, **bendout** initially transmits out port a, on MIDI channel 1.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to transmit pitch bend messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
(MIDI name)	Optional. The name of a MIDI output device may be used as the first argument to specify the port.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

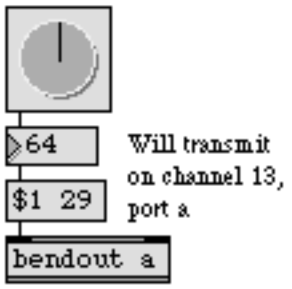
Output

(MIDI)	There are no outlets. The output is a MIDI pitch bend message transmitted directly to the object's MIDI output port.

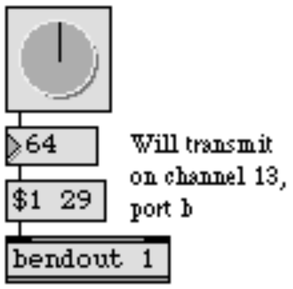
bendout

Transmit MIDI pitch bend messages

Examples



Letter argument transmits to only one port



Otherwise, number specifies both port and channel

See Also

bendin	Output received MIDI pitch bend messages
midiout	Transmit raw MIDI data
xbendout	Prepare extra precision MIDI pitch bend messages
xbendin	Interpret extra precision MIDI pitch bend messages
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified
Tutorial 16	More MIDI ins and outs

bondo

Synchronize a group of messages

Input

any message	In any inlet: The input is stored in the location corresponding to that inlet, and causes anything previously stored to be sent out its corresponding outlet. If no message has yet been received in a particular inlet, 0 is sent out of the corresponding outlet.
bang	In any inlet: Sends out all stored messages immediately.
set	In any inlet: The word set, followed by any message, stores the input in the location corresponding to that inlet without triggering any output.

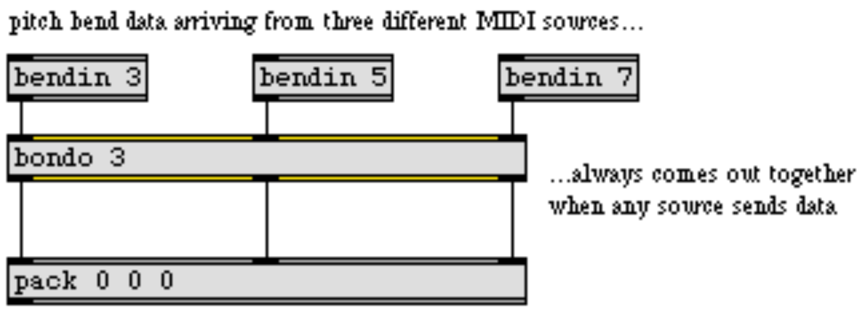
Arguments

int	Optional. The first argument specifies the number of inlets and outlets. The default number of inlets and outlets is 2. The second argument specifies a number of milliseconds to delay when a message is received before sending messages out the outlets.

Output

any message	Anything stored in an inlet is sent out the corresponding outlet numbers. Output is immediate if triggered by a bang. If output is triggered by a message, and a second argument has been typed in, output will be delayed by the number of milliseconds specified in the second argument.

Examples


pitch bend data arriving from three different MIDI sources...
bendin 3
bendin 5
bendin 7
bondo 3
...always comes out together when any source sends data
pack 0 0 0

bondo *can synchronize messages arriving from different sources*

See Also

buddy	Synchronize arriving data, output them together
onebang	Traffic control for bang messages
pack	Combine numbers and symbols into a list
thresh	Combine numbers into a list, when received close together

borax

Report current information about note-ons and note-offs

Input

int In left inlet: The number is the pitch value of a MIDI note-on message or note-off message (note-on with a velocity of 0). The pitch is paired with the velocity in the middle inlet. **borax** ignores note-on messages for pitches it is already holding, and ignores note-off messages for pitches that have already been turned off. If the note is not a duplicate, **borax** sends out the pitch and velocity values, as well as other information.

In middle inlet: The number is stored as the velocity, to be paired with pitch numbers received in the left inlet.

float In middle inlet: Converted to int.

list In left inlet: The second number is stored as the velocity, and the first number is used as the pitch, of a pitch-velocity pair. If the note is not a duplicate, **borax** sends out the pitch and velocity values, as well as other information.

delta In left inlet: Causes the *delta time* (the time elapsed since the last note-on) and the *delta count* (the number of delta times that have been reported) to be sent out.

bang In right inlet: Resets **borax** by sending note-offs for all notes currently being held, erasing the **borax** object's memory of all notes received, and setting its counters and its clock to 0.

Arguments

None.

Output

int Out left outlet: Each note-on received by **borax** is assigned a unique number, equal to the total count of note-ons received (since the last reset). That number is sent out when the note-on is received, and the same number is sent out when the note is turned off.

Out 2nd outlet: Each note is also assigned a unique voice number, equal to the lowest available number. (A voice becomes available when the note assigned to it is turned off.) That number is sent out when the note-on is received, and the same number is sent out when the note is turned off.

Out 3rd outlet: The number of notes being held by **borax** is sent out each time a note-on or a note-off is received.

Out 4th outlet: The pitch of the note-on or note-off is sent out.

Out 5th outlet: The velocity of the note-on or note-off is sent out.

Out 6th outlet: When a note-off is received, the total count of all completed notes (since the last reset) is sent out.

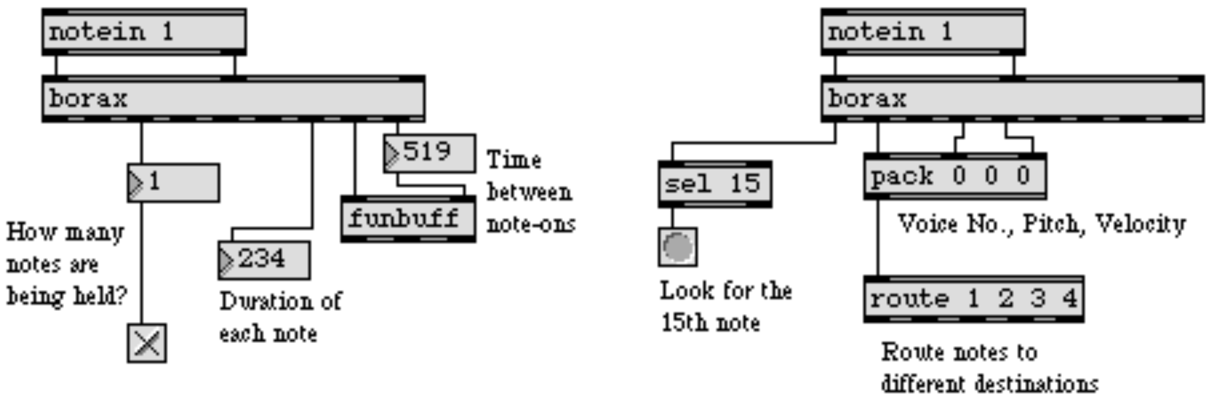
Out 7th outlet: When a note-off is received, the duration of that note, in milliseconds, is sent out.

Out 8th outlet: Each time a delta time is reported, the total count of delta times is sent out.

Out right outlet: When a note-on is received, the delta time is sent out (the time elapsed since the previous note-on, in milliseconds). A delta message in the left inlet causes the same output.

A bang received in the right inlet causes **borax** to provide note-offs for any notes it currently holds. These note-offs trigger the same outputs as if they had actually been received.

Examples



borax *provides extensive information about the notes passing through*

See Also

midiparse	Interpret raw MIDI data
poly	Allocate notes to different voices

bpatcher

Embed a visible subpatch inside a box

Input

anything — The number of inlets in a **bpatcher** object is determined by the number of **inlet** objects contained in its subpatch window. If the patch being used in a **bpatcher** contains **inlet** objects, they will appear in left-to-right correspondence as inlets in the **bpatcher** object's box.

offset — If the subpatch being used in the **bpatcher** contains a **thispatcher** object connected to one of its **inlet** objects, the view of the subpatch can be changed by an offset message received in the corresponding inlet of **bpatcher**. The word offset must be followed by two ints, specifying the number of pixels by which the upper left corner of the subpatch is to be offset horizontally and vertically within the bpatcher. In this way, a single **bpatcher** can be used to give different views of the subpatch. User interface objects in the subpatch that are partially outside the **bpatcher** object's box will redraw completely (even outside the bounds of the **bpatcher**) in response to messages received in their inlet. It is therefore advised that user interface objects in the subpatch be either completely inside or completely outside the **bpatcher** object's box.

border — If the subpatch being used in the **bpatcher** contains a **thispatcher** object connected to one of its **inlet** objects, the word border with any non-zero number in that inlet causes a black border to be drawn around the **bpatcher**. The message border 0 erases the border of the **bpatcher** (the default appearance).

(mouse) — When the window containing the **bpatcher** is locked (or the Command key on Macintosh or Control key on Windows is held down) and the mouse is clicked inside the **bpatcher** object's box, the gesture is handled by the patch inside the box.

If the Shift and Command keys on Macintosh or Shift and Control keys on Windows are held down while clicking on a **bpatcher**, dragging the mouse moves the upper-left corner of the visible part of the patch inside the box. The Assistance area of the patcher window shows the pixel values of the offset. If *Enable Drag-Scrolling* is unchecked in the **bpatcher** Inspector window, this feature is disabled.

If the Command and Option keys on Macintosh or Control and Alt keys on Windows are held down while clicking in a **bpatcher**, a pop-up menu allows you to open the original file of the patch contained inside the box in its own window, or change the patch currently contained inside the box in its own window.

Inspector

The behavior of a **bpatcher** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **bpatcher** object displays the **bpatcher** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **bpatcher** Inspector lets you set the following attributes:

Offset specifies the number of pixels by which the left upper corner of the picture is to be offset horizontally and vertically from the left upper corner of the **fpic** box. By default the left upper corner of the picture is located at the left upper corner of **fpic** (that is, with an offset of 0,0). This offset can be changed by entering new pixel values into the number boxes. The default is no offset (i.e. 0 horizontal, 0 vertical).

Use the *Offset* number boxes to specify the number of pixels by which the upper left corner of the subpatch is to be offset horizontally and vertically within the **bpatcher** object's display area. The default values are 0 for both horizontal and vertical offsets.

Checking the *Border* checkbox causes a black border to be drawn around the **bpatcher**. The default appearance is unchecked (no border).

The *Embed Patcher in Parent* checkbox allows you to embed the subpatch and save it as part of the main patch (just as with a **patcher** object) instead of the subpatch being saved in a separate file. The default is unchecked (the subpatch is saved as a separate file).

Checking the *Enable Drag-Scrolling* checkbox allows you move the upper-left corner of the visible part of the patch inside the box by holding down the Shift and Command keys on Macintosh or Shift and Control keys on Windows while clicking on a **bpatcher**, and dragging the mouse. The default value is unchecked (drag-scrolling is disabled).

The *Patcher File* option lets you choose a patcher file for the **bpatcher** to use by clicking on the *Open* button. The current file's name appears in the text box to the left of the button. You can also choose a file by typing its name in this box, or by dragging a file icon from the Finder into this box.

The *Arguments to Patcher* lets you input arguments to your patcher which will be saved along with the main patch.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

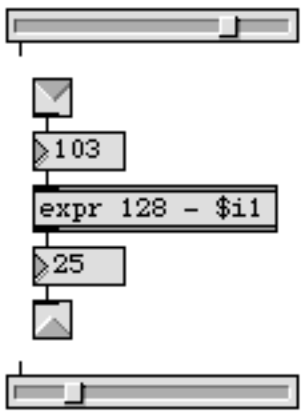
bpatcher

Embed a visible subpatch inside a box

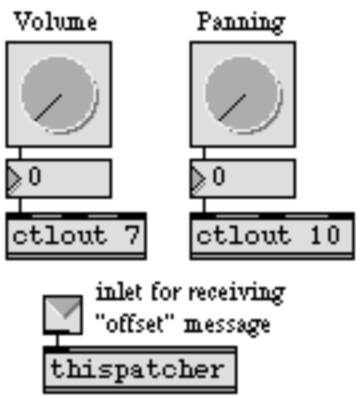
Output

If the patcher being used in a **bpatcher** contains **outlet** objects, they will appear in corresponding left-to-right order as outlets in the **bpatcher** object's box.

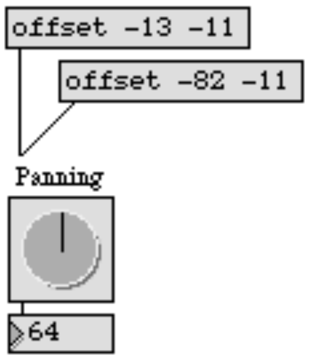
Examples



View the contents of a subpatcher



The contents of this patch can be windowed...



...using offset *messages to a small* **bpatcher** *containing it*

See Also

patcher	Create a subpatch within a patch
pcontrol	Open and close subwindows within a patcher
thispatcher	Send messages to a patcher
Tutorial 27	Your object
Tutorial 28	Your argument
Encapsulation	How much should a patch do?

bucket

Pass a number from outlet to outlet, out each one in turn

Input

int	The numbers currently stored in **bucket** are sent out, then each number is moved one outlet to the right and the new number is stored to be sent out the left outlet the next time a number is received.
float	Converted to int.
list	Only the first number in the list is used.
bang	All stored values are sent out, but their position is not shifted.
freeze	Suspends the **bucket** output, but new incoming numbers continue to shift the stored values internally.
thaw	Resumes **bucket** output.
roll	The word roll, followed by any number, causes **bucket** to use the value stored in its rightmost outlet as input; thus, it sends its output, shifts all stored values to the right, then stores the value which had been in the rightmost outlet in the leftmost outlet (as if it had been received in the inlet).
l2r	Sets **bucket** to shift its stored values from left to right (the default) whenever it receives a number in its inlet.
r2l	Sets **bucket** to shift its stored values from right to left whenever it receives a number in its inlet, placing the incoming number in the rightmost outlet.
set	The word set, followed by a number, sends that number out each outlet, and stores the number as the next value to be sent out each of its outlets.

Arguments

int	Optional. Sets the number of outlets. If there is no argument, there will be one outlet.

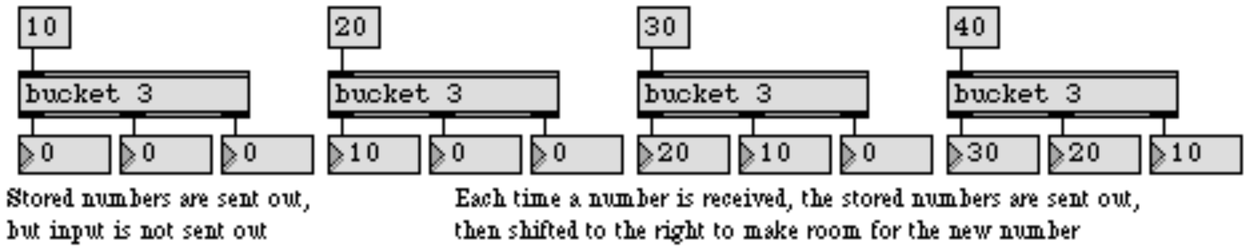
Output

int	When a number is received, it is not sent out immediately, but the numbers stored in **bucket** are sent out. The numbers are all moved one outlet to the right, and the newly received number is stored in the left position.

bucket

Pass a number from outlet to outlet, out each one in turn

Examples



10
bucket 3
0
0
0
20
bucket 3
10
0
0
30
bucket 3
20
10
0
40
bucket 3
30
20
10
Stored numbers are sent out, but input is not sent out
Each time a number is received, the stored numbers are sent out, then shifted to the right to make room for the new number

Numbers are passed from one outlet to another

See Also

cycle	Send a stream of data to individual outlets
decode	Send 1 or 0 out a specific outlet
gate	Pass the input out a specific outlet
spray	Distribute an integer to a numbered outlet

buddy

Synchronize arriving data, output them together

Input

any message	In any inlet: When data has been received in *all* its inlets, **buddy** sends the received messages out their corresponding outlets, then waits until data has arrived again in all inlets.
clear	In left inlet: Deletes all values stored in the inlets.
bang	In any inlet: Same as the number 0.

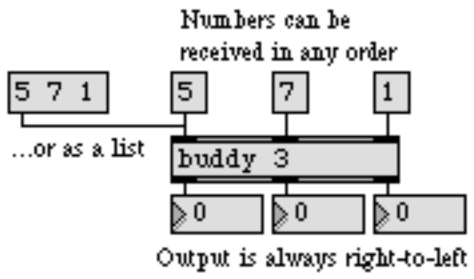
Arguments

int	Optional. Sets the number of inlets (and outlets). If there is no argument, there are two inlets and two outlets.

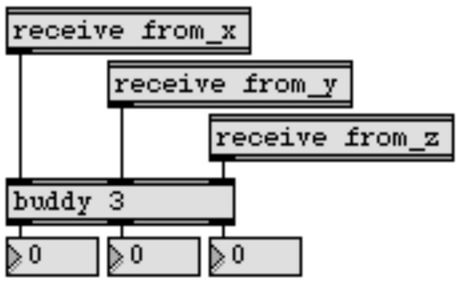
Output

any message	When a data has arrived in each inlet, it is sent out the outlets, in order from right to left.

Examples



Numbers can be received in any order
5 7 1
5
7
1
...or as a list
buddy 3
0
0
0
Output is always right-to-left



Output is synchronous, even if input is not synchronous

See Also

bondo	Synchronize a group of messages
onebang	Traffic control for bang messages
pack	Combine numbers and symbols into a list
swap	Reverse the sequential order of two numbers
thresh	Combine numbers into a list, when received close together
unpack	Break a list up into individual messages

button

Flash on any message, send a bang

Input

color	The word color, followed by a number from 0 to 15, sets the color of the center circle of the **button** to one of the object colors which are also available via the Color command in the Object menu. When **button** sends a bang, it always flashes with the color yellow.
any message	When any message is received in the inlet, **button** flashes briefly and bang is sent out the outlet. A mouse click on the **button** has the same effect.

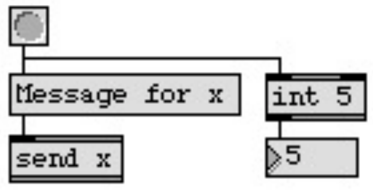
Arguments

None.

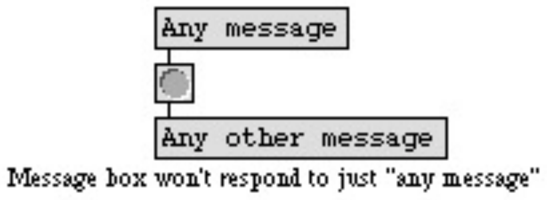
Output

bang	A mouse click or any message in the inlet causes **button** to flash and send out bang.

Examples



Triggers other messages and processes



Converts other messages to bang

See Also

bangbang	Send a bang to many places, in order
matrixcrtrl	Matrix-style switch control
pictctrl	Picture-based control
trigger	Send input to many places, in order
ubutton	Transparent button, sends a bang
Tutorial 2	bang means "Do it!"

capture

Store numbers to view or edit

Input

int, float, or symbol Numbers or symbols are stored in the order in which they are received.

list All numbers and/or symbols in the list are stored in order from first to last.

clear Erases the contents of a **capture** object.

count Sends the number of items collected since the last count message out the right outlet of the **capture** object.

dump Outputs the contents of the **capture** object, one item at a time, out the left outlet.

open Causes the window associated with the **capture** object to become visible. The window is also brought to the front. Double-clicking on the **capture** object in a locked patcher has the same effect.

wclose Closes the window associated with the **capture** object.

write The word write, followed by a symbol, saves the contents of the **capture** object into a text file, using the symbol as the filename. The file will be saved in the same folder as the Max application, unless the symbol is a pathname specifying some other folder (such as write "MyDisk:/Documents/Captured Data/outputfile"). The word write by itself causes a standard Save As dialog box to be opened, allowing you to name the file and save it in the desired folder.

Arguments

int Optional. The first argument sets a maximum number of items to store. If there is no argument, **capture** will store up to 512 items. Once the maximum has been exceeded, the earliest stored item is dropped as each new item is received.

a, x or m Optional. If the second argument is a, all items will be displayed in ASCII form in the editing window. If the second argument is x, all numbers will be displayed in hexadecimal form in the editing window.If the second argument is m, numbers less than 128 are displayed in decimal, and numbers greater than 128 are in hexadecimal. If there is no argument, all items are displayed in decimal.

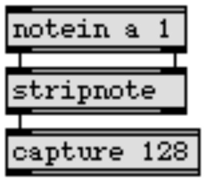
Output

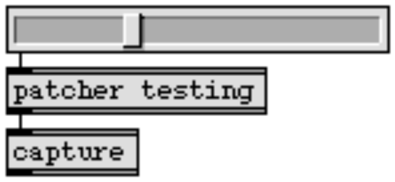
int, float, or symbol Out left outlet: The captured contents are sent out the left outlet, one at a time, in response to the dump message.

Double-clicking on **capture** (when the patcher window is locked) opens an editing window in which the stored numbers can be viewed and edited. Editing the window does not actually alter the contents of capture, but is useful for cutting and pasting values into a **table** or a separate file. (Although **capture** can continue to store items while the editing window is open, the editing window is not updated. It must be closed and reopened to view the newly stored items.)

int	Out right outlet: The number of items received since last count message was received is sent out the right outlet in response to a count message.

Examples





Collect numbers to paste into a table… *…or just to see what's been going on*

See Also

text	Format numbers as a text file
Debugging	Techniques for debugging patches
Tutorial 34	Managing raw MIDI data

cartopol

Cartesian to Polar coordinate conversion

Input

float	In left inlet: The real part of a frequency domain value to be converted into a polar coordinate pair consisting of amplitude and phase values.
	In right inlet: The imaginary part of a frequency domain value to be converted into a polar coordinate pair consisting of amplitude and phase values.
int	Converted to float.

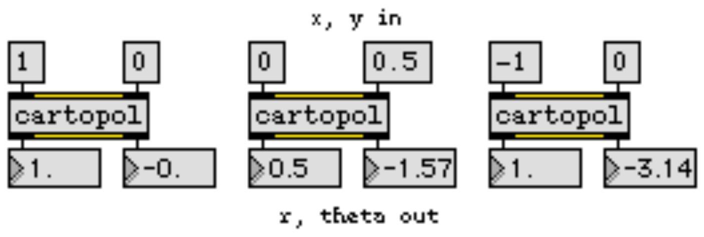
Arguments

None.

Output

float	Out left outlet: The magnitude (amplitude) of the frequency represented by the currently input.
	Out right outlet: The phase, expressed in radians, of the frequency represented by the current input. If only the left outlet is connected, the phase computation is not performed.

Examples



Convert Polar to Cartesian coordinates

See Also

atan2	Arc-tangent function (two variables)
lcd	Draw graphics in a patcher window
poltocar	Polar to Cartesian coordinate conversion
pow	Compute *x* to the power of *y*

change

Filter out repetitions of a number

Input

int or float — The number is sent out the outlet only if it is different from the currently stored value. Replaces the stored value.

set — The word set, followed by a number, replaces the stored value without triggering output.

mode — The word mode, followed by a +, causes **change** to send a 1 out its left outlet if the received number is greater than the previously received number. In this mode, **change** does nothing with any other input. The word mode, followed by a -, causes **change** to send out a -1 if the received number is less than the previously received number. In this mode, **change** does nothing with any other input. The word mode by itself returns **change** to its default mode of sending out received values that differ from the previously received input.

Arguments

int or float — Optional. Initial value for comparison to incoming numbers. If there is no argument, the initial value is 0.

symbol — Optional. A second argument may be + or -, causing **change** to behave as if it had received a mode + or mode - message. Subsequent mode messages can change this behavior.

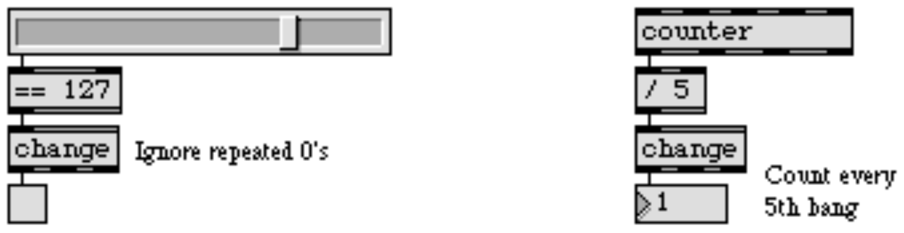
Output

int — Out left outlet: The number received in the inlet is sent out only if it is different from the stored value.

Out middle outlet: If the stored value is 0 and the input is not 0, 1 is sent out; otherwise nothing is sent out.

Out right outlet: If the stored value is not 0 and the input is 0, 1 is sent out; otherwise nothing is sent out.

Examples



Filter out undesirable repetitions

See Also

peak	If a number is greater than previous numbers, output it
togedge	Report a change in zero/non-zero values
trough	If a number is less than previous numbers, output it
!=	Compare two numbers, output 1 if they are not equal
Tutorial 15	Making decisions with comparisons

clip

Limit numbers within a certain range

Input

int or float — In left inlet: The number is sent out the outlet, constrained within the minimum and maximum limits specified by the arguments, inlets, or by a set message. If the number received is a float, it will be sent out as a float.

In middle inlet: Minimum limit for the range of the output.

In right inlet: Maximum limit for the range of the output.

list — Each number in the list is constrained within the minimum and maximum limits, and the constrained numbers are sent out as a list.

set — The word set, followed by two numbers, resets the minimum and maximum limits within which all numbers will be constrained before being sent out the outlet.

Arguments

int or float — Optional: The first number specifies a minimum limit and the second number specifies a maximum limit, within which all numbers will be constrained before being sent out the outlet. If only one argument is present, it is used as both the minimum and maximum limit. If no argument is present, the minimum and maximum limit is 0.

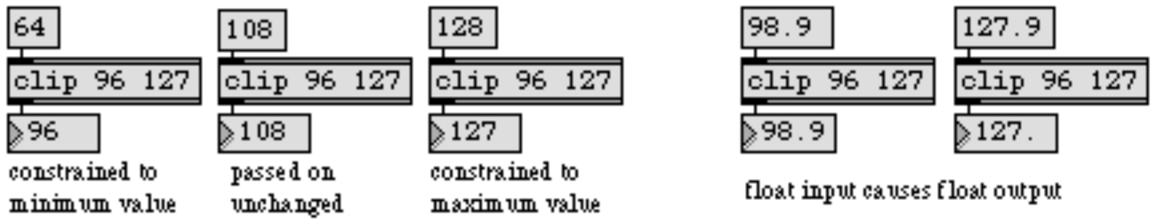
Output

int — When an int is received in the inlet, it is constrained within the specified minimum and maximum limits, then sent out the outlet. If the received number is less than the minimum limit, the minimum value is sent out; if the received number is greater than the maximum limit, the maximum value is sent out.

float — If the received number is a float, it is constrained within the specified minimum and maximum limits, then sent out the outlet as a float.

list — When a list is received in the inlet, each number is constrained within the specified minimum and maximum limits, and the numbers are sent out as a list.

Examples



Numbers are always kept within the specified range

See Also

maximum	Output the greatest in a list of numbers
minimum	Output the smallest in a list of numbers
split	Look for a range of numbers
<	*Is less than*, comparison of two numbers
<=	*Is less than or equal to*, comparison of two numbers
>	*Is greater than*, comparison of two numbers
>=	*Is greater than or equal to*, comparison of two numbers

clocker

Report elapsed time, at regular intervals

Input

int or float	In left inlet: Any non-zero number starts **clocker**. The time elapsed since **clocker** was started is sent out the outlet at regular intervals. 0 stops **clocker**. If clocker is already running when it receives a non-zero number, it continues reporting the elapsed time at regular intervals from that new point, but without resetting the clock time to 0. The **clocker** object's minimum interval time is 0.02 second.
	In right inlet: The number is the time interval, in milliseconds, at which **clocker** will report the elapsed time. A new number in the right inlet does not take effect until the next time output is sent.
bang	In left inlet: Starts **clocker**. If the **clocker** object is not running, a bang message will start the count. If the **clocker** object is running, a bang message will reset the count.
stop	In left inlet: Stops **clocker**.
clock	The word clock, followed by the name of an existing **setclock** object, sets the **clocker** to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets **clocker** back to using Max's regular millisecond clock.
reset	In left inlet: Resets the elapsed time to 0 without stopping or restarting the clock; **clocker** continues to report the new elapsed time at the same regular interval. This message is meaningless when the **clocker** is not running, since it always resets to 0 anyway when stopped.

Arguments

int	Optional. The first argument sets an initial value for the time interval at which **clocker** sends out its output. If there is no argument, the initial time interval is set to 5 milliseconds.

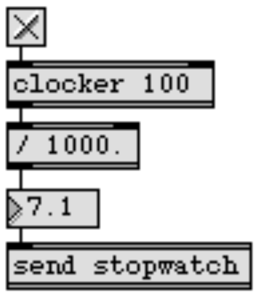
Output

int	The time elapsed, in milliseconds, since **clocker** was started. The first output is always 0, sent immediately each time **clocker** is started.

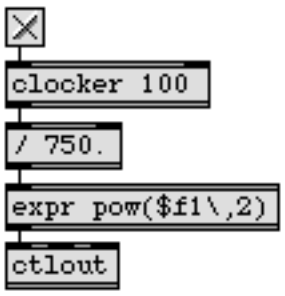
clocker

Report elapsed time, at regular intervals

Examples



Get the elapsed time



Generate numbers as a function of time

See Also

metro	Output a bang message at regular intervals
setclock	Control the clock speed of timing objects remotely
tempo	Output numbers at a metronomic tempo
timer	Report elapsed time between two events
Tutorial 31	Using timers

closebang

Send a bang *when patcher window is closed*

Input

There are no inlets. Output occurs when the patcher window is closed.

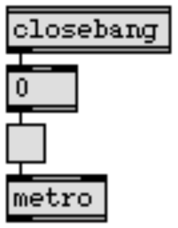
Arguments

None.

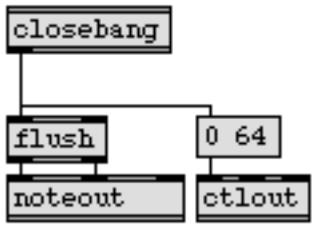
Output

bang	Sent automatically when the patcher window is closed.

Examples



Stop a process when window is about to be closed



...or turn off held notes and sustain pedal

See Also

active	Send 1 when patcher window is active, 0 when inactive
button	Flash on any message, send a bang
loadbang	Send a bang automatically when patch is loaded
Tutorial 40	Automatic actions

coll

Store and edit a collection of different messages

Input

list	The first number is used as the *address* (the storage location within **coll**) at which to store the remaining items in the list (**coll** can store a list of up to 250 items). The address will always be stored as an int.
int or float	The number refers to the address of a message stored in **coll**. If a message is stored at that address, the stored message is sent out the 1st outlet.
bang	Same effect as the next message.
(Get Info…)	A **coll** object can be set to save its contents as part of the patch that contains it. When the patcher window is unlocked, select the **coll** object, choose Get Info… from the Object menu, and check *Save coll with patcher.*
assoc	The word assoc, followed by a symbol and a number, *associates* the symbol with the address specified by the number, provided that the number address already exists. From then on, any reference to that symbol will be interpreted by **coll** as a reference to the number address. Each number address can have only one symbol associated with it, except 0, which cannot have an associated symbol. (Note: If the symbol was already being used as an address, or was already associated with a number address, the message that was stored at that address is removed.)
clear	Erases everything from the collection.
deassoc	The word deassoc, followed by a symbol and a number, removes the association between the symbol and the number address. The symbol no longer has any meaning to **coll**.
delete	Functions similarly to the word remove, except that if the specified address is a number, all addresses of a greater number are decremented by 1.
dump	Sends *all* of the stored addresses out the 2nd outlet and all of the stored messages out the 1st outlet, in the order in which they are stored. A bang is sent out the 4th outlet when the dump is completed.
end	Sets the pointer (used by the goto, next, and prev messages) to the last address in the **coll**.
filetype	The word filetype, followed by a symbol, sets the file types which can be read and written into the **coll** object. File types are specified are specified using the standard four-letter type code combination (e.g. filetype ffoo). The message filetype with no arguments restores the default file behavior—either Max binary or text file formats. File types are mapped to filename extensions on Windows based on the messages to max contained in the file max-fileformats.txt in the init folder, which is loaded on startup. If you are defining your own filetype, you may want to include your own text file in the init folder in order to specify a mapping between an extension and your four-letter type code.

flags	Normally, the contents of **coll** are not saved as part of the patch when the patcher window is closed. The message flags 1 0 sets the **coll** object to save its contents as part of the patcher that contains it. The message flags 0 0 causes the contents of the **coll** not to be saved with the patcher that contains it.
goto	The word goto, followed by a number or a symbol, sets a pointer at the address specified by the number or symbol. If no such address exists, the pointer is set at the beginning of the collection. The pointer is set at the beginning of the collection initially, by default.
insert	The word insert, followed by a number and a message, inserts the message at the address specified by the number, incrementing all equal or greater addresses by 1 if necessary.
length	Counts the number of messages contained in **coll** and sends the number out the 1st outlet. This message works well in conjunction with the **grab** object.
max	Determines the *maximum* single numerical value (i.e. not a list or symbol) stored in the **coll** and sends the number out the 1st outlet. This message works well in conjunction with the **grab** object.
merge	The word merge, followed by an address and a message, appends its message at the end of the message already stored at that address. If the address does not yet exist, it is created.
min	Determines the *minimum* single numerical value (i.e. not a list or symbol) stored in the **coll** and sends the number out the 1st outlet. This works well in conjunction with the **grab** object.
next	Sends the address pointed to by the pointer out the 3rd outlet, and sends the message stored at that address out the 1st outlet, then sets the pointer to the next address. If the address is a symbol rather than a number, 0 is sent out the 3rd outlet. If the pointer is currently at the last address in the collection, it *wraps around* to the first address. (Note: Number addresses are stored in ascending order. Symbol addresses are stored in the order in which they were added to the collection, after all of the number addresses.) If the message received immediately prior to next was prev, next sends out the value stored at the address one greater than the one that was just sent out.
nstore	The word nstore, followed by a number and a symbol (or a symbol and a number), followed by any other message, stores the message at the specified number address in the **coll**, with the specified symbol associated. (This has the same effect as storing the message at an int address, then using the assoc message to associate a symbol with that number.)
nsub	The word nsub, followed by an address, an item number, and another number or symbol, replaces one item stored at the address. (Example: nsub pgms 4 7 puts the

number 7 in place of the 4th item of the message stored at the address pgms.) Number values and symbols can both be substituted in this manner.

nth	The word nth, followed by an address and a number, gets the *nth* item (specified by the number) from the message at that address, and sends it out the 1st outlet. (Example: nth pgms 4 outputs the 4th item in the message stored at the address named pgms.)
open	Causes a text edit window associated with the **coll** object to become visible. The window is also brought to the front.
prev	Causes the same output as the word next, but the pointer is then decremented rather than incremented. If the pointer is currently at the first address in the collection, it *wraps around* to the last address. If the message received immediately prior to prev was next, prev sends out the value stored at the address one less than the one that was just sent out.
read	The word read with no arguments puts up a standard Open Document dialog box for choosing a file to load into **coll**. If read is followed by a symbol filename argument, the named file is located and loaded into **coll**.
readagain	Loads in the contents of the most recently read file. If no prior read or readagain message has been received by the **coll**, readagain is treated as a read message, and an Open Document dialog box is displayed.
refer	The word refer, followed by the name of another **coll** object, changes the **coll** receiving the message to refer to the data in the named **coll** object.

In addition to reading messages in from another file and storing messages via the inlet, one can also enter messages in **coll** by typing. Double-clicking with the mouse on the **coll** object displays the contents as text in an editing window which the user can modify.

In order to edit a collection by hand or read in from another file, it is essential to know the correct text format for the contents of a **coll** object. Each message is stored in the **coll** object on a separate line. The format of each line is as follows: the address (an int or a symbol), any symbols associated with that address (if the address is an int), a comma (to separate the address from the data it contains), the data (anything), and a semicolon to indicate the end of each line. In a line such as

```
3 reset, set 4.7;
```

3 is the number of the address, reset is a symbol associated with that address, and the message it contains is set 4.7.

Here is how we would store the numbers 100, 200, 300, and 400 with the addresses 1, 2, 3, and 4.

```
1, 100;
2, 200;
3, 300;
4, 400;
```

remove The word remove, followed by a number or a symbol, removes that address and its contents from the collection.

renumber Makes the numbers associated with the data in the **coll** object consecutive and increasing. The argument to the renumber message specifies the starting number address for the data. Here's a before and after example for **coll** sent the message renumber 1.

Before	*After*
4, apple;	1, apple;
6, banana;	2, banana;
3, cherry;	3, cherry;
9, durian;	4, durian;

sort The sort message takes two arguments. If the first argument is -1, the items in the **coll** are sorted in ascending order. If the first argument is 1, the items in the **coll** are sorted in descending order.

The second argument specifies what is used to sort the contents of the **coll**. If the second argument is -1, the index (or symbol) associated with the data is used. If the second argument is not present or 0, the first item in the data is used. If the second argument is 1 or greater, the second (or greater) item in the data is used.

store The word store, followed by some symbol (usually a word), followed by a message, stores the message at an address named by the symbol. (Example: store triad 0 4 7 will store the list 0 4 7 at an address named triad.)

sub Same as nsub, except that the message stored at the specified address is sent out after the item has been substituted.

swap The swap message takes two symbols or two numbers as addresses, and exchanges the data associated with each address. For example, if the **coll** contains

```
1, 400;
2, 700;
```

swap 1 2 would change the **coll** to

```
1, 700;
2, 400;
```

subsym	Changes the symbol associated with data. The first argument to subsym is the new symbol to use, and the second argument is the symbol associator to replace. For instance, if the **coll** contains
	
	jill, 40 50 60;
	
	subsym jack jill will change the **coll** to
	
	jack, 40 50 60;

symbol	The symbol refers to the address of a message stored in **coll**. If a message is stored at the address named by the symbol, the message is sent out the 1st outlet. The symbol may, but need not necessarily, be preceded by the word symbol.
wclose	Closes the window associated with the **coll** object.
write	Calls up the standard Save As dialog box, enabling the user to save the contents of **coll** as a separate file. If the word write is followed by a symbol, the contents of the **coll** are saved immediately in a file, using the symbol as the filename.
writeagain	Saves the contents of the **coll** into the most recently written file. If no prior write or writeagain message has been received by the **coll**, writeagain is treated as a write message, and a Save As dialog box is opened.

Inspector

The behavior of a **coll** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **coll** object displays the **coll** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

Checking *Save coll with patcher* sets the **coll** object to save its contents as part of the patch that contains it.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

any symbol	Optional. Name of a file to be read into **coll** automatically when the patch is loaded. The information in the file must be in the correct format in order to be read in by **coll**. All **coll** objects which share the same name always share the same contents. You can use the file name as an identifier for the purpose of sharing data

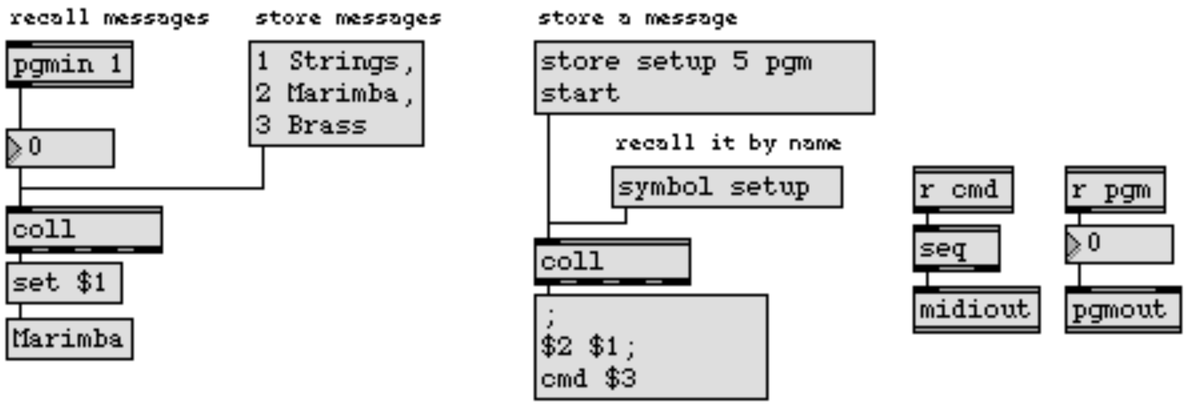
between multiple **coll** objects, without there needing to be an actual file with the specified name.

An optional second argument will cause the **coll** object not to search for a file with the named symbol.

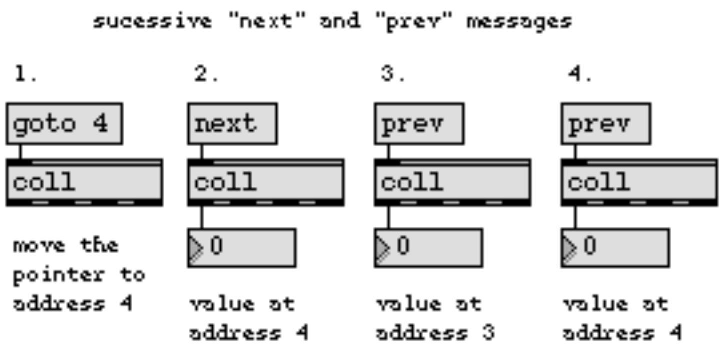
Output

anything	Messages stored in **coll** are sent out the 1st outlet. If the message consists of only a single symbol, it will be preceded by the word symbol when it is sent out.
int	Out 1st outlet: The number of messages contained in **coll** is sent out in response to the length message.
int or symbol	Out 2nd outlet: The address is sent out whenever a message out the 1st outlet is triggered by bang, dump, next, prev, or sub.
bang	Out 3rd outlet: Sent out when **coll** has finished loading in or writing a file of data.
	Out 4th outlet: Sent out when **coll** has finished sending all of the stored addresses and messages in order out the 1st and 2nd outlets in response to a dump message.

Examples



Complex messages can be recalled with a single number or word



Results for successive next *and* prev *messages*

See Also

bag	Store a collection of numbers
table	Store and graphically edit an array of numbers
funbuff	Store x,y pairs of numbers together
Tutorial 37	Data structures
Data Structures	Ways of storing data in Max

colorpicker

Select a color using a modal dialog

The **colorpicker** object uses an Operating System color picker dialog that lets you choose a color to be output as a Max RGB color. On the Mac OS, the Color Picker dialog that lets you choose colors in several different color spaces—red-green-blue (RGB), hue-saturation-value (HSV), web-safe colors, and the nostalgia-inducing crayon mode. On Windows, you are presented with a standard color picker dialog, including a selection of basic colors, custom colors, a color swatch and numerical input for red-green-blue (RGB), hue-saturation-luminance (HSL)

Input

(mouse)	Double-clicking the object opens the Color Picker dialog box. If the patcher is unlocked, hold down the Command key on Macintosh or the Control key on Windows while double-clicking to open the dialog.
bang	Same as double-clicking the object.
list	A list of three numbers between 0 and 255 specifies the RGB color components of the default color which initially appears in the Color Picker dialog box when it is opened.
setprompt	The word setprompt, followed by a text string, sets the Color Picker dialog box text label. This change will take effect the next time the dialog box is opened.

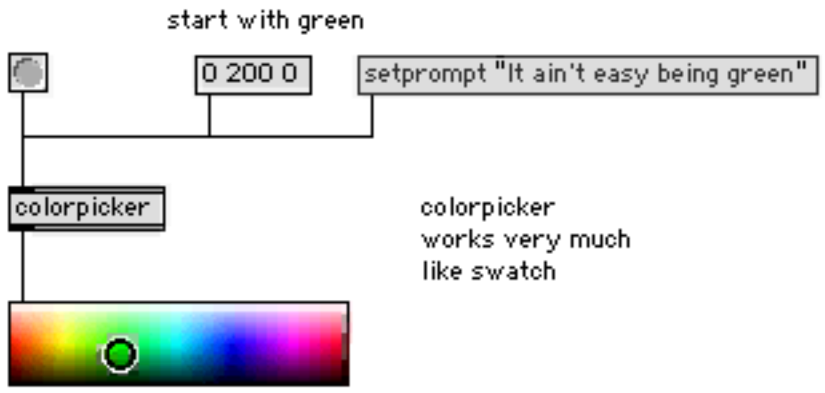
Arguments

None.

Output

list	After you open the Color Picker dialog box and make a selection, clicking on the OK button will send a list of the RGB equivalents of the color you selected out the outlet. If you click the Cancel button, no messages are sent.

Examples



Display a color, or retrieve selected RGB color values

colorpicker

Select a color using a modal dialog

See Also

panel	Colored background area
swatch	Color swatch for RGB color selection and display

comment

Explanatory note or label

Input

anything The **comment** object has no inlets and receives no input. Text is typed directly into the **comment** box when the patcher window is in Edit mode. When the patcher window is locked, the outline of the **comment** box disappears, and only the text is shown. The appearance of a **comment** can be modified by changing the font and by resizing its box. Note: If you want to include carriage returns in your text, use the Inspector to set two-byte compatibility mode.

The font and size of a **comment** can be changed with the Font menu.

Inspector

The appearance of a **comment** object can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **comment** object displays the **comment** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **comment** Inspector lets you set the following attributes:

You can set a comment to display text in languages such as Japanese or Chinese that use a two-byte character representation system by checking the *Two-byte Compatible* option (the default is unchecked). Checking the two-byte compatibility option will also allow you to include carriage returns in comment boxes.

The *Color* option lets you use a swatch color picker or RGB values used to display the comment text. The default text color is black (0 0 0).

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

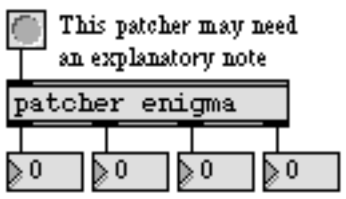
Output

A **comment** has no outlets, sends no output, and does not affect the functioning of the patch.

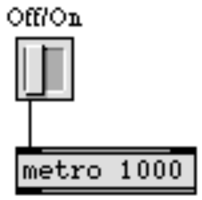
comment

Explanatory note or label

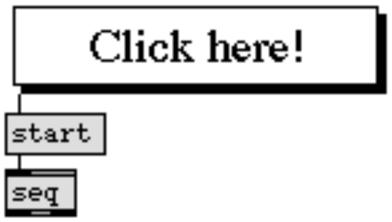
Examples



Elucidate



Label



Make functional (covered with a **ubutton***)*

See Also

ubutton	Transparent button, sends a bang
Tutorial 5	toggle and comment

conformpath

Convert paths of one pathtype and/or pathstyle to another

Input

any symbol	A file name or path as a symbol. The **conformpath** object converts paths of one *pathstyle* (i.e., file paths that use colons or slashes as separators) and/or *pathtype* (paths that are absolute, relative, boot volume-relative, or Cycling 74 folder-relative) to another. It provides a superset of the functionality of the **absolutepath** and **relativepath** objects.
pathstyle	The word pathstyle, followed by a word that specifies a pathstyle, will conform the output pathname to the chosen styles. The possible styles are:

colon	The *colon* style will use colons as separators when passing paths between objects. This style was used in Max versions 4.2 and earlier on Macintoshes Note: Since the native Macintosh pathstyle is the same as the colon path style, there is no native_mac pathstyle.
max	(default) The *max* style will use whatever style the currently running version of Max uses to pass paths between objects.
native	The *native* style will use whatever format is used by the currently running operating system to specify paths. Note: When working with native paths, only absolute paths will be valid for the operating system.
native_win	The *native_win* style will use native Windows OS format (i.e., backslashes as separators) to specify paths. Note: The use of the native_win style paths is not advised except for display purposes—In MaxMSP, the backslash character is used as an escape character and could lead to problems if used in conjunction with message boxes, **sprintf**, **coll**, and other objects which parse text into atoms.
slash	The *slash* style will use slashes as separators when passing paths between objects.

pathtype	The word pathtype, followed by a word that specifies a pathtype, will conform the output pathname to the chosen type. The possible types are:

absolute	The *absolute* type will output the absolute pathname of the file or folder as a symbol.
boot	The *boot* type will output the pathname of the file or folder relative to the boot volume as a symbol. If the file is not relative to the

	boot file, the **conformpath** object will send a zero out the right outlet and send the output path out the left outlet unchanged.
C74	The *C74* type will output the pathname of the file or folder relative to the Cycling 74 folder as a symbol. If the file is not relative to the Cycling 74 folder, the **conformpath** object will send a zero out the right outlet and send the output path out the left outlet unchanged.
ignore	(default) The *ignore* type will perform no path type conversion.
relative	The *relative* type will output the pathname of the file or folder relative to the Max application folder as a symbol. If the file is not relative to the Max application folder, the **conformpath** object will send a zero out the right outlet and send the output path out the left outlet unchanged.

Arguments

symbol Optional. An optional symbol argument specifies the pathtype to be used as output. The possible pathtype arguments are:

absolute	Specifies the output of the absolute pathname of the file or folder as a symbol.
boot	Specifies the output of the pathname of the file or folder relative to the boot volume as a symbol.
C74	Specifies the output of the pathname of the file or folder relative to the Cycling 74 folder as a symbol.
ignore	Specifies that no pathtype conversion is performed.
relative	Specifies the output of the pathname of the file or folder relative to the Max application folder as a symbol.

symbol Optional. An optional symbol argument specifies the pathstyle to be used as output. The possible pathstyle arguments are:

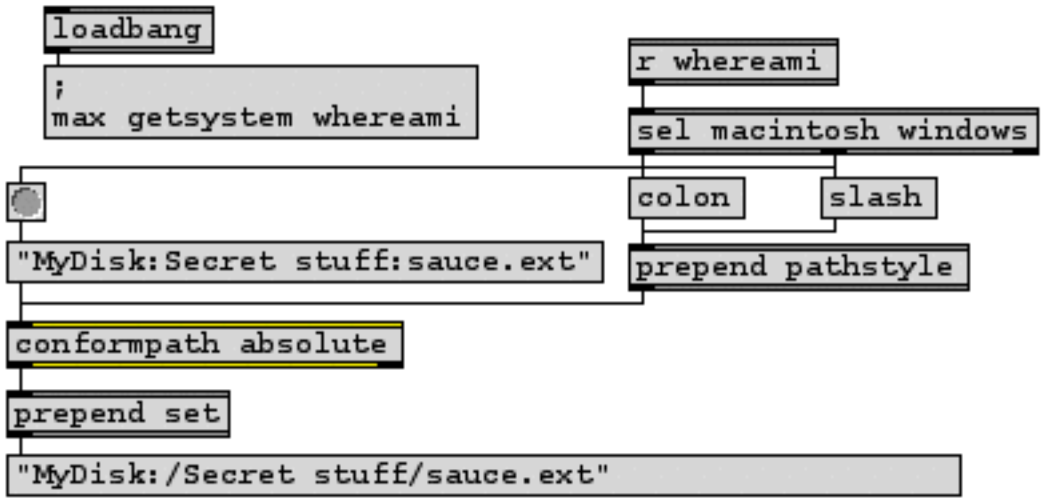
colon	Specifies that the colon pathstyle is used for output (See description in Input section for more details).
max	Specifies that the max pathstyle is used for output (See description in Input section for more details).
native	Specifies that the native pathstyle is used for output (See description in Input section for more details).

native_win	Specifies that the native_win pathstyle is used for output (See description in Input section for more details). Note: The use of the native_win style paths is not advised except for display purposes.
slash	Specifies that the slash pathstyle is used for output (See description in Input section for more details).

Output

symbol	The pathname of the folder or file conformed to the specified pathstyle and/or pathtype.
int	Out right outlet: If the input file or folder is conformed to specified pathtype and/or pathtype, the output is 1. if the filepath cannot be conformed (e.g., if the file is not relative to a requested path type), the output is 0.

Examples



Use the getplatform *message to Max to automatically conform file pathnames across platforms*

See Also

absolutepath	Convert a file name to an absolute path
opendialog	Open a dialog to ask for a file or folder
relativepath	Convert an absolute to a relative path
savedialog	Open a dialog to ask for a filename for saving
strippath	Get a filename from a full pathname

cos

Input

float	Input to a cosine function.
bang	In left inlet: Calculates the hyperbolic cosine of the number currently stored. If there is no argument, **cos** initially holds 0.

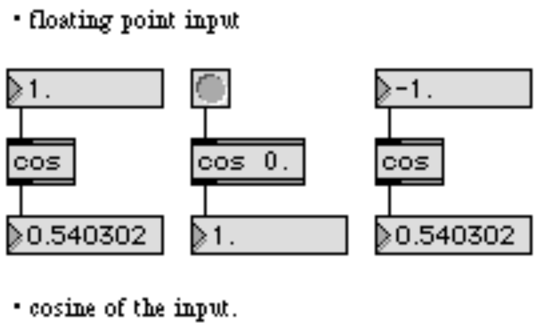
Arguments

float or int	Optional. Sets the initial value for the cosine function.

Output

float	The cosine of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

cosh

Input

float or int	Input to a hyperbolic cosine function.
bang	In left inlet: Calculates the hyperbolic cosine of the number currently stored. If there is no argument, **cosh** initially holds 0.

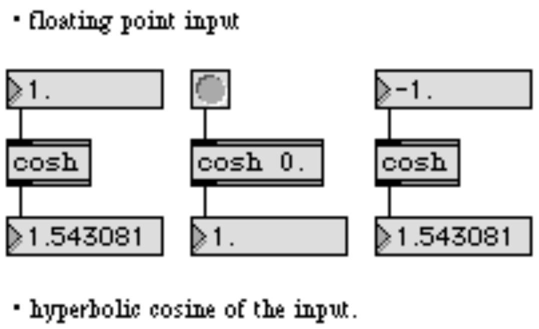
Arguments

float or int	Optional. Sets the initial value for the hyperbolic cosine function.

Output

float or int	The hyperbolic cosine of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

counter

Count the bang *messages received, output the count*

Input

bang — In left inlet: Sends out the current count of the bang messages received in the left inlet.

In left-middle inlet: Changes the direction of the count.

In middle inlet: Resets the count to its specified minimum value, which will be sent out the next time a bang is received in the left inlet.

In right-middle inlet: Resets the count to its specified minimum value, and sends out that value immediately.

In right inlet: Resets the count to its specified maximum value, which is sent out immediately.

int — In left inlet: Same effect as bang.

In left-middle inlet: Sets the direction of the count. 0 causes **counter** to count up, 1 causes it to count down, and 2 causes it to count up and down.

In middle inlet: The number sets the counter to a new value, to be sent out the next time a bang is received in the left inlet. If the number is less than the current minimum value, the minimum will be reset to that number. If the number is greater than the current maximum value, the counter will be set to that number, but the maximum value actually remains the same and the minimum is set equal to the maximum.

In middle-right inlet: The number sets the counter to a new value and sends it out immediately. If the number is less than the current minimum value, the minimum will be reset to that number. If the number is greater than the current maximum value, the number is sent out, but the maximum value actually remains the same and the minimum is set equal to the maximum.

In right inlet: Resets the maximum value sent out by **counter**. If the number is less than the current minimum, the maximum is equal to the minimum. If the minimum is subsequently changed to a value below the maximum value you input, the **counter** objects retains the correct maximum value it received through this inlet. Unlike a bang message, an int in this inlet does not cause the **counter** object to output anything.

float — In left inlet: Same effect as bang.

float — In all other inlets: Converted to int.

carrybang — In left inlet: Causes **counter** to send a bang out the right-middle outlet when the count is going upward and reaches its maximum limit, and causes **counter** to send a bang out the left-middle outlet when the count is going downward and reaches

	its minimum limit. (By default, counter sends out the number 1 in those situations, instead of bang.) The state of the carrybang message is saved along with the patcher it is used in, and this behavior can also be set using the Inspector.
carryint	In left inlet: Undoes the effect of a previously received carrybang message. Resets the counter to send the numbers 1 and 0 out the left-middle and right-middle outlets (instead of bang) to signal when the counter reaches and leaves its minimum and maximum values. The state of the carryint message is saved along with the patcher it is used in, and this behavior can also be set using the Inspector.
dec	In left inlet: Decrements the counter (downward) and sends out the new value, regardless of the direction in which the object has been set to count ordinarily.
down	In left inlet: Sets the **counter** to count in a downward direction.
goto	In left inlet: Same effect as set.
inc	In left inlet: Increments the counter (upward) and sends out the new value, regardless of the direction in which the object has been set to count ordinarily.
jam	In left inlet: The word jam, followed by a number, sets the counter to that number and sends the number out immediately. If the number is outside the minimum and maximum count range, this message is ignored.
min	In left inlet: The word min followed by a number, resets the minimum value of **counter** to that number, and causes the **counter** object to set itself to that number and output immediately. If the number is greater than the current maximum value, the minimum is set equal to the maximum.
max	In left inlet: The word max followed by a number, resets the maximum value of **counter** to that number. If the number is less than the current minimum value, the maximum is considered to be equal to the minimum, although the actual maximum value you set is stored inside the **counter** object.
next	In left inlet: Same as bang.
set	In left inlet: The word set, followed by a number, sets the counter to that number, which will be sent out the next time a bang is received in the left inlet.
setmin	In left inlet: The word setmin, followed by a number, sets the **counter** object's minimum count without affecting its current count value or causing any output.
up	In left inlet: Sets the **counter** to count in an upward direction.
updown	In left inlet: Sets the **counter** object's direction so that it counts upward until it reaches the specified maximum, then counts down until it reaches the specified minimum, then up, then down, and so on.

Inspector

The behavior of an **counter** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **counter** object displays the **counter** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The *Underflow/Carry Mode* attribute provides two options correspond to the carrybang and carryint messages described above. Sending 1 or 0 out outlets 2 and 3 is the default mode.

The *Reset Minimum Mode* attribute lets you choose between temporarily overriding the min count (the default behavior). Sending an int to the third and fourth inlets of the counter object will cause it to perform in the manner described in the Input section above. The *Change the Min count permanently* option provides back-compatibility with the **counter** object distributed with Max 3.x and earlier. In this mode, sending an int to inlets 3 and 4 will change the min count instead of just resetting it temporarily (which causes the fourth inlet to behave exactly as thought the min message were sent to the **counter** object).

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

int Optional. If there is only one argument, it sets an initial maximum count value for **counter**. If there are two arguments, the first number sets an initial minimum value, and the second number sets an initial maximum value. If there are three arguments, the first number specifies the direction of the count, the second number is the minimum, and the third number is the maximum. If there are no arguments, the direction is up, the minimum is 0, and the maximum is 2,147,483,647 (the largest possible 32-bit signed integer).

Output

int Out left outlet: When bang, next, inc, dec, or a number is received in the left inlet, the current count is sent out, within the minimum and maximum limits specified. If the direction of the count is both up and down, the count is *folded back* in the other direction when it reaches the specified limits. If the count is in only one direction, up or down, the count is *wrapped around* to the opposite extreme when it reaches its limit.

When the direction is up, or up and down, **counter**, begins counting from the specified minimum value. When the direction is down, **counter** begins from the maximum value.

Out left-middle outlet: When the count is moving downward and reaches the minimum limit, the number 1 is sent out. When the count leaves the minimum limit, 0 is sent out.

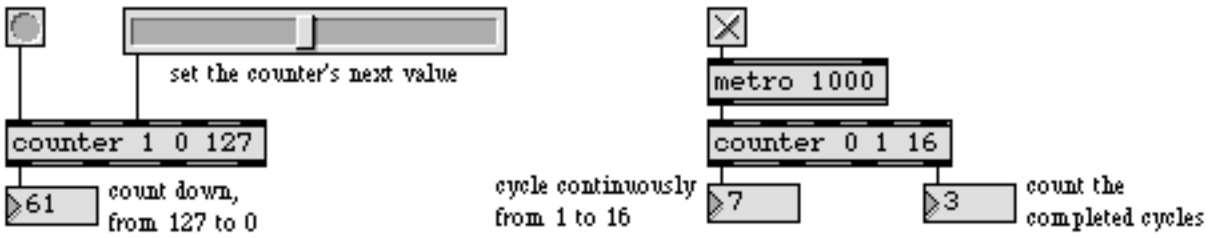
Out right-middle outlet: When the count is moving upward and reaches the maximum limit, the number 1 is sent out. When the count leaves the maximum limit, 0 is sent out.

Out right outlet: An additional count is kept of the number of times **counter** reaches its maximum limit. Each time the maximum is reached, that count is sent out.

bang Out left-middle outlet: If a carrybang message has been received in the left inlet, then when the count is moving downward and reaches the minimum limit, a bang is sent out (instead of the number 1 which is sent out by default). When the count leaves the minimum limit, nothing is sent out.

Out right-middle outlet: If a carrybang message has been received in the left inlet, then when the count is moving upward and reaches the maximum limit, a bang is sent out (instead of the number 1 which is sent out by default). When the count leaves the maximum limit, nothing is sent out.

Examples


set the counter's next value
counter 1 0 127
61
count down, from 127 to 0
metro 1000
counter 0 1 16
cycle continuously from 1 to 16
7
3
count the completed cycles

Keep track of how many events have occurred, or create a continuous loop

See Also

tempo	Output numbers at a metronomic tempo
Tutorial 31	Using timers
Loops	Using loops to perform repeated operations

ctlin

Output received MIDI control values

Input

(MIDI)	**ctlin** receives its input from a MIDI control change message received from a MIDI input device.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming control messages. The word port is optional and may be omitted.
set	The word set, followed by a number from 0 to 127, specifies a single controller number to be paid attention to by **ctlin**. This message is appropriate only if a specific controller number was originally typed in as an argument; it is ignored by **ctlin** if no controller number argument was originally typed in.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
(mouse)	Double-clicking on a **ctlin** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

a-z	Optional. Specifies a single port from which to receive incoming control messages. If there is no letter present as an argument, **ctlin** can receive from all ports.
(MIDI name)	Optional. The name of a MIDI input device may be used as the first argument to specify the port.
int	Following the (optional) port argument, the next argument is a single controller number to be recognized by **ctlin**. If there is no controller number, or if the argument is a negative number, **ctlin** recognizes *all* controller numbers. If a single controller number *is* specified in the argument, the outlet which normally sends the controller number is unnecessary, and is not created.

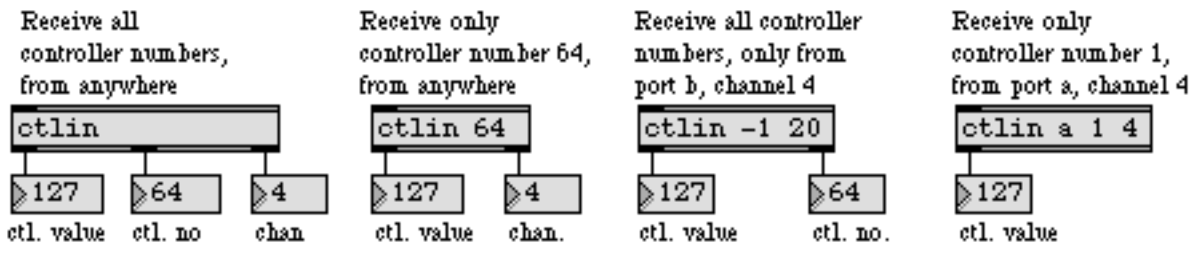
Following the controller number argument is a single channel number on which to receive control messages. If the channel argument is not present, **ctlin** receives control messages on all channels. In order for this argument to be used, a controller number argument *must* precede it. To specify a channel number without specifying a controller number, use -1 for the controller number.

If a single channel number is specified as an argument, the outlet which normally sends the channel number is unnecessary, and is not created. If a port has been specified with a letter argument, channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range. If no port argument is present, a channel number can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

Output

int	Out left outlet: The number is the control value of an incoming MIDI control change message.
	If a specific controller number is not specified as an argument, the controller number is sent out the 2nd outlet.
	If a specific channel number is not included in the argument, the channel number is sent out an additional, right, outlet.

Examples


Receive all controller numbers, from anywhere
ctlin
127
64
4
ctl. value
ctl. no
chan
Receive only controller number 64, from anywhere
ctlin 64
127
4
ctl. value
chan.
Receive all controller numbers, only from port b, channel 4
ctlin -1 20
127
64
ctl. value
ctl. no.
Receive only controller number 1, from port a, channel 4
ctlin a 1 4
127
ctl. value

Control messages can be filtered in a variety of ways

See Also

bendin	Output received MIDI pitch bend values
ctlout	Transmit MIDI control messages
midiin	Output received raw MIDI data
notein	Output received MIDI note messages
rtin	Output received MIDI real time messages
xbendin	Interpret extra precision MIDI pitch bend messages
MIDI	MIDI software protocol
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified
Tutorial 16	More MIDI ins and outs

ctlout

Input

int	In left inlet: The number is used as the control value, and **ctlout** transmits a MIDI control change message. Numbers are limited between 0 and 127.
	In middle inlet: The number is stored as the controller number of the control change messages transmitted by **ctlout**. Numbers are limited between 0 and 127.
	In right inlet: The number is stored as the channel number on which to transmit the control messages.
float	Converted to int.
list	In left inlet: The first number is the control value, the second the controller number, and the third the channel number. **ctlout** transmits a MIDI control change message using these values.
enable	The message enable 0 disables the object, causing it not to transmit MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	In left inlet: The word port, followed by a letter a-z or the name of a MIDI output port or device, specifies the port used to transmit MIDI control messages. The word port is optional and can be omitted.
(mouse)	Double-clicking on a **ctlout** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

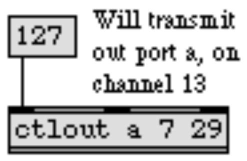
a-z	Optional. Specifies the port for transmitting MIDI control messages. If there is no argument, **ctlout** initially transmits out port a, on channel 1. When a port is specified by a letter argument, channel numbers greater than 16 received in the right inlet will be *wrapped around* to stay within the 1-16 range.
(MIDI name)	Optional. The name of a MIDI output device may be used as the first argument to specify the port.
int	Following the (optional) port argument, the next argument is an initial value for the controller number to be used in control messages transmitted by **ctlout**. Controller numbers are automatically limited between 0 and 127. If there is no controller number specified, the initial controller number is 1.
	Following the controller number argument is an initial value for the channel number on which to transmit control messages. If the channel argument is not present, **ctlout** initially transmits control messages on channel 1. In order for this argument to be used, a controller number argument *must* precede it.

If a port has been specified with a letter argument, channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range. If no port argument is present, the channel number specifies both the port and the channel. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

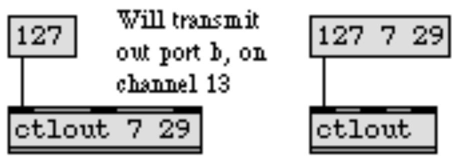
Output

(MIDI) There are no outlets. The output is a MIDI control message transmitted directly to the object's MIDI output port.

Examples



Letter argument transmits to only one port



Otherwise, number specifies both port and channel

See Also

bendout	Transmit MIDI pitch bend messages
ctlin	Output received MIDI control values
midiout	Transmit raw MIDI data
noteout	Transmit MIDI note messages
xbendout	Format extra precision MIDI pitch bend messages
MIDI	MIDI overview and specification
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified
Tutorial 16	More MIDI ins and outs

cycle

Send a stream of data to individual outlets

Input

anything The stream of ints, floats, or symbols to be directed to successive outlets.

set The word set, followed by a number, specifies an outlet to which the next input should be directed, if in cycle mode. Outlets are numbered beginning with 0; if an outlet number is specified that does not actually exist, the message is ignored. (This message has no effect when **cycle** is in event-sensitive mode, in which case each message is always sent out beginning at the leftmost outlet.)

thresh The word thresh, followed by a number, sets the output mode, in the same way as the second typed-in argument. If the number is non-zero, **cycle** will detect separate "events" and restart at the leftmost outlet whenever a new event occurs. If the number is 0, each number received will be directed to the next outlet in the cycle.

Arguments

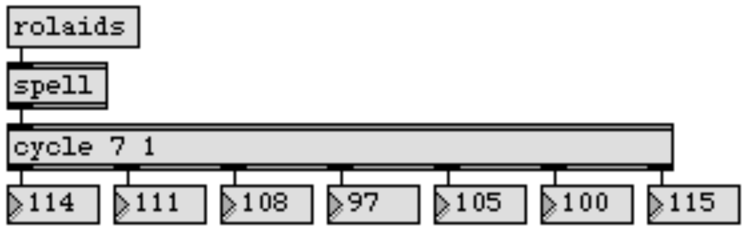
int Optional. The first argument determines the number of outlets. If there is no argument, there will be one outlet. The second argument sets the output mode. If it is non-zero, **cycle** detects separate "events" and restarts at the leftmost outlet when a new event occurs. Examples of separate events include messages with delays between them, and messages triggered by successive mouse clicks or MIDI events. A stream of items separated by commas in a message box is considered a single event. If this argument is not present or is 0, the values cycle through all the outlets, regardless of whether they are attached to separate events or not.

Output

anything Out any outlet: In *cycle* mode, each successive int, float, or symbol received, either separately or as part of a list, is directed to an outlet to the right of the previous number. When the cycle reaches the rightmost outlet, the next number is sent out the left outlet.

In *event-sensitive* mode, any int, float, or symbol which is a new event restarts the output at the left outlet.

Examples



Using **cycle** *to get ASCII relief*

See Also

bucket	Pass a number from outlet to outlet, out each one in turn
counter	Count the bang messages received, output the count
spell	Convert input to ASCII codes
spray	Distribute an integer to a numbered outlet

date

Report current date and time

Input

date	Outputs the current date as a list (month/day/year) out the left outlet.
ticks	Outputs the current value of Ticks (the number of 1/60ths of a second since system startup) out the right outlet.
time	Outputs the current time as a list (military hours/minutes/seconds) out the middle outlet.

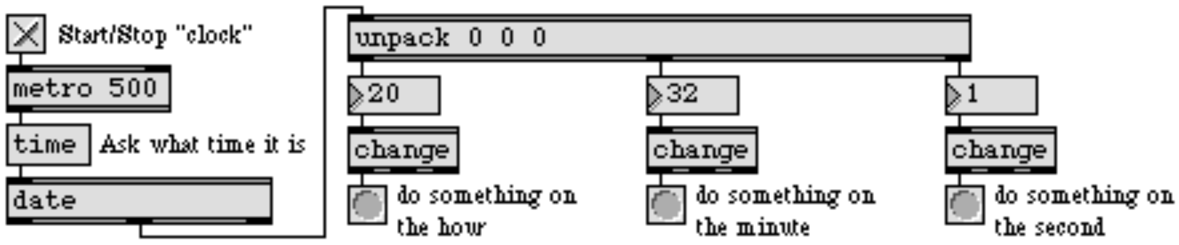
Arguments

None.

Output

list	Out left outlet: When the date message is received, **date** sends the current date as a list.
list	Out middle outlet: When the time message is received, **date** sends the current time as a list.
int	Out right outlet: When the ticks message is received, **date** sends the current value of Ticks.

Examples



For pieces which change slowly, **date** *can be used as a clock to trigger events*

See Also

clocker	Report elapsed time, at regular intervals
timer	Report elapsed time between two events

decide

Choose randomly between on and off (1 and 0)

Input

bang	In left inlet: Causes a randomly chosen output of 1 or 0.
int	In left inlet: Same as bang.
	In right inlet: A given "seed" number causes a specific (reproducible) sequence of pseudo-random 0 and 1 outputs to occur. The number 0 uses the time elapsed since system startup (an unpredictable value) as the seed, ensuring an unpredictable sequence of 0 and 1 outputs.

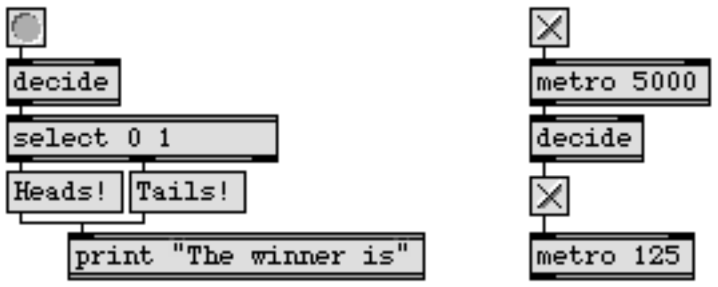
Arguments

int	Optional. Sets a "seed" value to cause a specific (reproducible) sequence of pseudo-random 0 and 1 outputs to occur. If there is no argument, the time elapsed since system startup (an unpredictable value) is used as the seed, ensuring an unpredictable sequence of 0 and 1 outputs.

Output

int	A 1 or a 0, chosen at random. With certain seed values, the output may seem at first to follow a "non-random" pattern, but over the course of many iterations the sequence becomes unpredictable and the balance between 1 and 0 becomes even.

Examples



Simulate a coin toss; switch randomly between on and off

See Also

drunk	Output random numbers in a moving range
random	Generate a random number
toggle	Switch between on and off (1 and 0)
urn	Generate random numbers without duplicates

decode

Send 1 or 0 out a specific outlet

decode acts as a hierarchical switchboard. The right inlet is the master switch, which can turn off (send 0 out) all outlets. The middle inlet is a submaster switch, which can turn on (send 1 out) all outlets, *provided* they have not all been turned off by the master switch. The left inlet can turn on one of the outlets exclusively, provided neither the submaster switch nor the master switch is active.

Input

int In left inlet: The number specifies an outlet out to turn on, turning off all other outlets. (Whenever an outlet is turned on that was previously turned off, a 1 is sent out. Conversely, whenever an enabled outlet is disabled, a 0 is sent out.) The outlets are referred to by number, beginning with 0 on the left, and numbers received in the left inlet are automatically limited between 0 and the number of outlets minus 1.

In middle inlet: Any number other than 0 enables all disabled outlets (sends a 1 out them), unless all outlets are disabled. When 0 is received, **decode** turns off all outlets except the one that had previously been on.

In right inlet: Any number other than 0 disables all enabled outlets (sends a 0 out them). Once all outlets have been disabled in this manner, no outlet can be enabled until a 0 is received in the right inlet. When a 0 is received, **decode** re-enables all outlets that it had just disabled.

float Converted to int.

Arguments

int Optional. Sets the number of outlets. The default is one outlet.

float Converted to int.

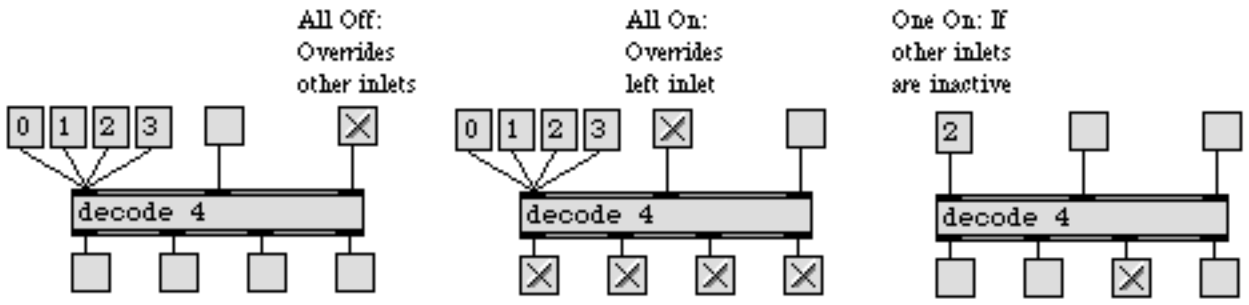
Output

int When an outlet is enabled that was previously disabled, a 1 is sent out that outlet. When an outlet is disabled that was previously enabled, a 0 is sent out that outlet. The left outlet is initially enabled.

decode

Send 1 or 0 out a specific outlet

Examples


All Off: Overrides other inlets
All On: Overrides left inlet
One On: If other inlets are inactive
0
1
2
3
decode 4
decode 4
2
decode 4

decode *is a hierarchical on/off switch*

See Also

bucket	Pass a number from outlet to outlet, out each one in turn
gate	Pass the input out a specific outlet
toggle	Switch between on and off (1 and 0)

defer

De-prioritize a message

Input

anything	If the message received in the inlet was triggered by a MIDI object (such as **notein**) or a timing object (such as **metro** or **seq**), and the Overdrive option is on, Max normally gives the message priority over activities that are not so critical in their timing (such as printing in the Max window). The **defer** object removes that special priority from a message, allowing it to be superseded by messages for which precise timing is more critical. This is useful for de-prioritizing time-consuming messages which may interfere with musical rhythm, or for messages to objects that may not function well with Overdrive on.

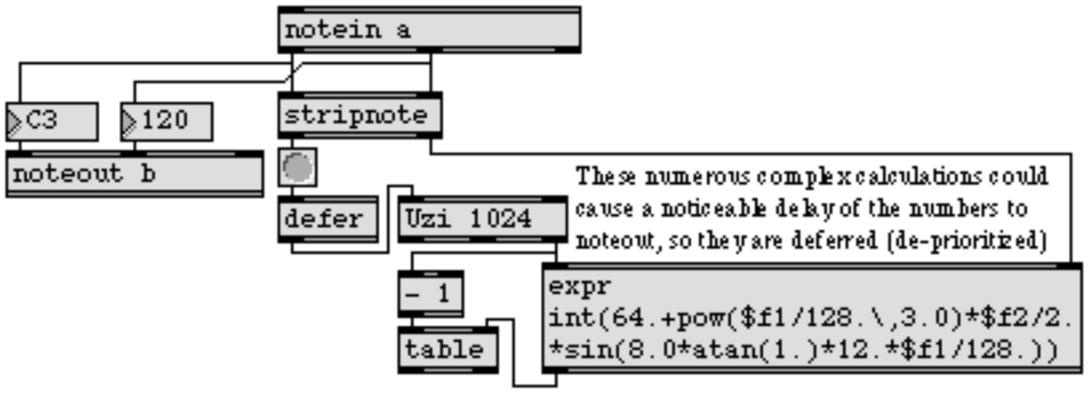
Arguments

None.

Output

anything	Same as the input.

Examples


notein a
C3
120
stripnote
noteout b
defer
Uzi 1024
These numerous complex calculations could cause a noticeable delay of the numbers to noteout, so they are deferred (de-prioritized)
- 1
table
expr int(64.+pow($f1/128.\,3.0)*$f2/2.*sin(8.0*atan(1.)*12.*$f1/128.))

Overdrive's priority given to MIDI or timing messages can be overridden with **defer**

See Also

uzi	Send a specific number of bang messages

delay / del

Delay a bang
before passing it on

Input

bang	In left inlet: A bang is delayed a certain number of milliseconds before being sent out the outlet.
stop	In left inlet: Stops **delay** from outputting the bang it is currently delaying.
int or float	In left inlet: Sets the number of milliseconds to delay a bang, then triggers the bang to be delayed.
int or float	In right inlet: The number is stored as the number of milliseconds to delay a bang received in the left inlet. A number received in the right inlet changes the delay time of the next bang received—it does not modify the time of a bang currently being delayed.

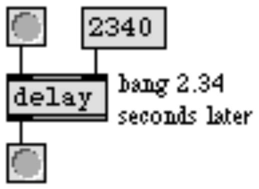
Arguments

int or float	Sets an initial value for the number of milliseconds to delay a bang received in the left inlet. If there is no argument, the initial value is 0.

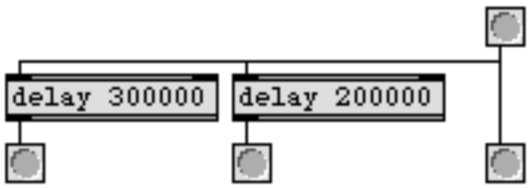
Output

bang	A bang received in the left inlet is delayed by the number of milliseconds specified by the right inlet, then is sent out the outlet. Only one bang at a time can be delayed by **delay**. If a bang is already in **delay** when a new bang is received in the left inlet, the first bang is forgotten.

Examples



Bang is delayed for a certain time



Can be used to send triggers at specific times

See Also

pipe	Delay numbers or lists
Tutorial 22	Delay lines

Input

int After a record message has been received, all numbers received are treated as parameters of a note event.

In left inlet: The delta time (delay), in milliseconds, since the previous recorded event. This denotes the "inter-onset interval —the time between the beginnings of notes—which effectively determines the rhythm in which the events are recorded. This need not necessarily be the true time in which they occur; **detonate** believes any (non-negative) delta time it receives.

In 2nd inlet: The number is treated as the key number (pitch) of the note. If no key number has ever been received, 60 is used by default.

In 3rd inlet: The velocity of the note. If the velocity is 0—indicating a note-off—the event will be treated as the end of an earlier note-on the same key, and will determine the duration of that earlier note. If no velocity number has ever been received, it is 64 by default.

In 4th inlet: In lieu of a note-off message, a note duration can be supplied as part of the note-on event. If no duration value has ever been received, and no note-off event is received to end the note, a duration of 10 milliseconds is used by default.

In 5th inlet: The number of a track on which to record the note event. Overdub recording is not possible with **detonate**, but each recorded note can be tagged with a track number for storing separate tracks of notes internally. If no track number has ever been received, notes are recorded on track 1.

In 6th inlet: The MIDI channel of the note. If no channel has ever been specified, notes are recorded on channel 1.

In 7th inlet: An "extra" number, which can be used for any purpose, attached to the note event. This number can be used to provide an additional event parameter, or to serve as a control value in sync with the note. If no number has ever been received in this inlet, it is recorded as 0 by default.

In right inlet: A second "extra" number.

When **detonate** receives a number in the left inlet while recording, it treats the number as the inter-onset interval (the time elapsed since the previous event), combines it with the numbers most recently received in the other inlets, and records them together as a note event. As with most Max objects, the numbers received in the other inlets are stored for use in subsequent note events triggered by the receipt of a number in the leftmost inlet.

When **detonate** has received a follow message (see below), a subsequent number in the 2nd inlet is treated as the key number (pitch) of a note. If the number is the same as the pitch of the current note in the score (or a nearby note), the information recorded for that note—except for the delta time—is sent out.

When **detonate** is neither recording nor following, a number in the left inlet has the same effect as the nth message (see below).

float	Converted to int.
list	The first number in the list is used as the delta time, and the other numbers are treated as if they had been received in the other inlets, respectively from left to right.
start	Begins playing back the score, by simply sending out the first delta time. Once playback of the score has been started, next messages can be used to send out the next event information.
next	Once playback of the score has been started with a start message, next sends out the event information (except the delta time) for the current note in the score, then sends out the delta time for the next note. That delta time can in turn be used as a delay time before sending another next message to detonate. When next is received on the last note of the score, there is no note following that one, so a unique value of -1 is sent out the left outlet to signal the end of the score. If a next message is received while the score is not being played back, **detonate** simply prints the message not playing in the Max window.
nth	The word nth, followed by a number, sends out the note information of the event in the score indicated by the number. (Events are numbered beginning with 0.) In place of the delta time for the event, the (cumulative) starting time of the event is sent out the left outlet.
clear	Erases the contents of **detonate**.
follow	Causes **detonate** to behave like a score reader, comparing incoming pitch information to the events stored in its score. When a key number is received in the 2nd (pitch) inlet, and it is the same as the pitch of the current note in the score, **detonate** sends out the information recorded for that event—except for the delta time—and then moves ahead to the next note event.
followat	The word followat, followed by a pitch, a velocity, and a MIDI channel number, causes **detonate** to look for a note event with those attributes in its stored score. If such a note is found, **detonate** commences score-following from the next event onward. If not, it simply prints detonate: note not found in the Max window.
record	In left inlet: Begins recording numbers coming in the inlets, treating them as parameters of note events to be recorded in a graphic score. The onset of an event is recorded each time a number is received in the left inlet.

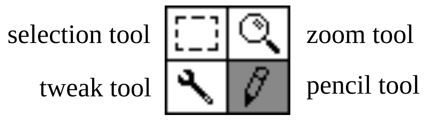
startat	The word startat, followed by a pitch, a velocity, and a MIDI channel number, causes **detonate** to look for a note event with those attributes in its stored score. If such a note is found, **detonate** sends out the delta time of the next event, and a subsequent next message will refer to that next event. If no such note is found, **detonate** simply prints detonate: note not found in the Max window.
stop	Stops **detonate** from recording, playing, or following. It is not necessary to stop detonate before switching directly between record, start, and follow.
mute	Permits the selective muting of note events that meet specific criteria. The word mute must be followed by an event parameter number, a parameter value, and a value of 1 or 0 signifying "mute" or "unmute". Event parameters are numbered beginning at 0 for delta time, 1 for pitch, etc. For example, the message mute 4 10 1 mutes notes on MIDI channel 10 (channel is parameter 4), preventing their note information from being sent out; those notes can later be unmuted by the message mute 4 10 0.
unmute	The word unmute, followed by an event parameter number and a parameter value, undoes an earlier mute of the same criterion. For example, unmute 4 10 has the same meaning as mute 4 10 0.
unmuteall	Undoes the effects of all previous mute messages.
params	The word params, followed by three numbers, modifies the score-following behavior of **detonate** for cases when the received pitch does not match the pitch of the current note in the score. The first number tells **detonate** how many errors to tolerate before moving ahead in the score. The second number tells how many milliseconds to move ahead in the score when too many errors have occurred. The third number, if non-zero, tells **detonate** to treat a received pitch that is an octave too high or too low as if it were a match. For example, the message params 3 1000 1 means to allow three successive errors (with octave displacements considered to be a match) before moving ahead one second in the score and resuming. By default, **detonate** allows 2 errors before moving ahead 200 milliseconds, and does not consider octave pitch displacements to be a match for the stored note.
write	Opens a dialog for saving the contents of **detonate** as a standard MIDI file. The word write may optionally be followed by up to two numbers. If the first number is non-zero, the file will be saved with time represented in milliseconds rather than as bars, beats, and ticks in a certain tempo. If the number is 0 or not present, the file is saved as beats. The second number indicates the MIDI file format: 0 (all notes on a single track) o multi-track format, using the track parameter to separate the notes). The contents of **detonate** are also saved as part of the patch, when the patch is saved.
read	The word read by itself opens a dialog for loading in a standard MIDI file as contents of the **detonate** score. If read is followed by the name of a MIDI file in Max's search path, that file is read in directly without opening a dialog box. The read

message can also be followed by a number which—if non-zero—causes the time values in the file to be interpreted as milliseconds rather than as bars, beats and ticks at a certain tempo. If the number is 0 or not present, the times are read as bars and beats.

export Same as write.

import Same as read.

(mouse) Double-clicking on **detonate** in a locked patcher opens an editor window to display a graphic representation of the note events. The editor window can show the event information in various ways, and contains a small palette of tools for editing the notes or entering new notes.



You can draw new notes with the pencil tool. The starting time of note events is always represented on the x axis of the graph. The default parameters of the drawn notes are shown in (and can be changed by dragging upon) the number boxes at the top of the editor window. You can change the meaning ascribed to the y axis, and to the length of the drawn note, by clicking on the icons to the left of the parameter names. By default the y axis is pitch and the horizontal length of the note shows its duration.

You can select existing notes with the selection tool, and drag them either vertically (by clicking in the middle of a note) or horizontally (by clicking on the left side of note). Dragging on the right side of a note enables you to lengthen or shorten it. The parameters of selected notes can also be changed with the number boxes at the top of the editor window.

The tweak tool works the same as the selection tool, but allows for finer resolution dragging adjustments. Clicking on the graph with the zoom tool enlarges that area of the graph for more precise editing. Option-clicking on Macintosh or Alt-clicking on Windows on the graph with the zoom tool zooms back out.

Arguments

symbol Supplies a name to be shown in the title bar of **detonate's** graphic editor window. Any **detonate** objects with the same name argument will share the same event data. They will also share event data with any **edetonate** timeline editor that has the same name.

Output

When **detonate** receives a start message or a startat message in the left inlet, it sends out the delta time of its starting note event (or of the note after the found note, in the case of startat). After that, each time **detonate** receives a next message, it sends out all the other note data for that event, and the delta time of the next event, progressing through the score. Thus, the numbers coming out the left outlet can be used to control the playback rhythm, by delaying for the specified time and then triggering the next next message.

When **detonate** receives an nth message (or receives a number, while stopped) in the left inlet, it uses that information as an index number (starting at index number 0 for the first note event) and sends out all note data for the indexed event. Instead of sending the note's delta time out the left outlet, however, it sends the start time of the note—the total time since the beginning of the score.

After **detonate** has received a follow or followat message in the left inlet, if a number is received in the 2nd inlet that matches the pitch of the current note in the score (or one of the two notes immediately after it), all the data for the matched note is sent out, except for the delta time.

int Out left outlet: When a start, startat, or subsequent next message is received in the left inlet, the delta time of the next note event is sent out. When the last event in the score is played by a next message, there is no note following that one, so a unique delta time of -1 is sent out to signal that the last note has been played.

When an nth message is received in the left inlet (or an int if **detonate** is stopped), the starting time of the specified note is sent out.

Out 2nd outlet: In response to an nth message, or an int while **detonate** is stopped, or a next message while playing back, or a matched pitch while following, the pitch of the note is sent out.

Out 3rd outlet: The velocity of the note.

Out 4th outlet: The duration of the note.

Out 5th outlet: The MIDI channel of the note.

Out 6th outlet: The track number of the note.

Out 7th outlet: An extra value associated with the note.

Out right outlet: A second extra value associated with the note.

Inspector

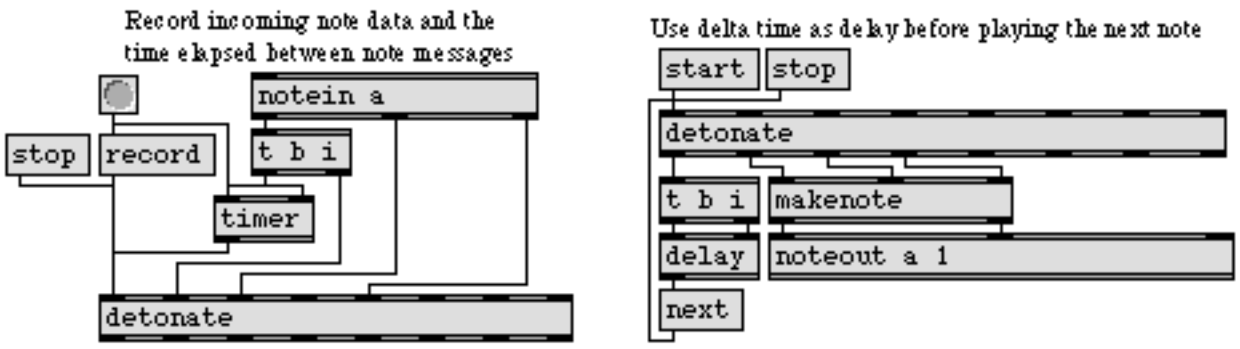
You can change the depiction of the **detonate** object's parameters (corresponding to the object's inlets) by reassigning the way each parameter is shown. The menu at the top of the inspector lets you select which of the eight parameters (numbered 0 through 7) will be displayed in the Display.

You can change the name of the parameter using the *Parameter Name* field. The default names are Time, Pitch, Vel, Dur, Chan, X1 and X2. The *Display Mode* menu lets you set how the parameter is displayed in the **detonate** graphic editor. Parameters can be displayed along the X-axis, Y-axis, Length (along the x-axis) or as a Number. Setting the menu to No Display, naturally causes the parameter not to be displayed.

Each parameter's Minimum Value and Maximum Value can be set using the fields with those names. The *Default Value* sets the value which will be used for that parameter in notes where it is left unspecified.

Graph Interval affects the view only if the parameter is displayed on the y axis; it controls how often numbers will be shown along the y axis (every 12 semitones in the above example). *Default Scaling* is a factor that determines the default zoom of the axis on which the parameter is being displayed. 1 is maximum zoom, and larger numbers are successively smaller scales. The start time (the leftmost parameter) is an exceptional case because it can only be displayed on the x axis; so, for that parameter Graph Interval and Default Scaling refer only to the x axis.The *Display MIDI Note Numbers* checkbox can be used to display values on the y axis as MIDI notes instead of decimal numbers only for parameter 1 (pitch); this option is disabled for all other parameters.

Examples


Record incoming note data and the time elapsed between note messages
notein a
stop
record
t b i
timer
detonate
Use delta time as delay before playing the next note
start
stop
detonate
t b i
makenote
delay
noteout a 1
next

Note events are recorded with a delta time, which can be used to play notes back in rhythm

See Also

follow	Compare a live performance to a recorded performance
seq	Sequencer for recording and playing MIDI
timeline	Time-based score of Max messages
Detonate	Graphic editing of a MIDI sequence
Sequencing	Recording and playing back MIDI performances

dial

Output numbers by moving a dial onscreen

Input

int	The number received in the inlet is displayed graphically by **dial**, and is passed out its outlet. Optionally, **dial** can multiply the number by some amount and add an offset to it before sending it out the outlet. The **dial** will also send out numbers in response to clicking or dragging on it directly with the mouse.
float	Converted to int.
bang	Sends out the number currently stored in **dial**.
brgb	The word brgb, followed by three numbers between 0 and 255, sets the background color of the dial in RGB format. The default is gray (221 221 221).
color	The word color, followed by a number from 0 to 15, sets the color of the center circle of the **dial** to one of the object colors which are also available via the Color command in the Object menu.
frgb	The word brgb, followed by three numbers between 0 and 255, sets the color of the center dial in RGB format. The default is light gray (170 170 170).
min	The word min, followed by a number, sets value that will be added to the **dial** object's value before it is sent out the outlet. The default is 0.
mult	The word mult followed by a number, specifies a multiplier value. The **dial** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default value is 1.
rgb2	The word rgb2, followed by three numbers between 0 and 255, sets the center dial (Foreground) of the dial in RGB format. The default is dark grey (120 120 120).
rgb3	The word rgb3, followed by three numbers between 0 and 255, sets the highlighted border around the center dial in RGB format. The default is off-white (225 225 225).
rgb4	The word rgb4, followed by three numbers between 0 and 255, sets the color of the dial indicator (needle) in RGB format. The default is black (0 0 0).
rgb5	The word rgb5, followed by three numbers between 0 and 255, sets the color of the frame/border of the dial in RGB format. The default is black (0 0 0).
set	The word set, followed by a number, changes the displayed value of the **dial**, without triggering output.

size The word size, followed by a number, sets the range of the **dial** object. The default value is 128. Setting the size to 1 disables the **dial** visually (since it can only display one value). Any specified size less than 1 will be set to 2.

Inspector

The behavior of a **dial** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **dial** object displays the **dial** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **dial** Inspector lets you enter a *Dial Range* value. Numbers received in the inlet are automatically limited between 0 and the number 1 less than the specified range value. The default range value is 128. You can specify an *Offset* value which will be added to the number, after multiplication. The default offset value is 0. The **dial** Inspector also lets you specify a *Multiplier*. The **dial** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default multiplier value is 1.

The *Colors* options let you use a swatch color picker or RGB values to specify the colors used for the **dial** object's display. *Foreground* sets the color for the face of the dial (default 170 170 170), and *Background* sets the color for the square area in which the dial appears (default 221 221 221). The *Frame* attribute sets color for the border around the **dial** object's square frame (default 0 0 0). The "lit" and "shaded" edges of the dial are set by the *Highlight* (default 255 255 255) and *Shadow* (default 120 120 120) attributes. The *Needle* attribute sets the color of the position indicator for the **dial** (default 0 0 0).

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

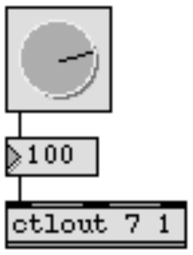
Output

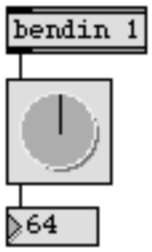
int Numbers received in the inlet, or produced by clicking or dragging on **dial** with the mouse, are first multiplied by the multiplier, then have the offset added to them, then are sent out the outlet.

dial

Output numbers by moving a dial onscreen

Examples





Produce output by dragging onscreen... *or use to display numbers passing through*

See Also

hslider	Output numbers by moving a slider onscreen
pictctrl	Picture-based control
pictslider	Picture-based slider
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

dialog

Open a dialog box for text entry

Input

symbol	In left inlet: The word symbol, followed by any word, opens a dialog box prompting the user to enter text. The word following symbol is shown as the default text. If you want more than one word to appear as the default text, you must enclose the words in double quotes.
bang	In left inlet: Opens the dialog box with the previous text displayed as the default.
int	In left inlet: Same as symbol.
	In right inlet: The number 0 sets **dialog** so that whatever the user types into the dialog box is sent out as a symbol preceded by the word symbol. A nonzero number sets **dialog** so that the typed-in text is sent out exactly as is if it begins with a word, or preceded by the word list if it begins with a number. If no number is received, it is considered 0 by default.

Arguments

anything	Optional. Sets the prompt which will appear above the text entry box in the dialog window.

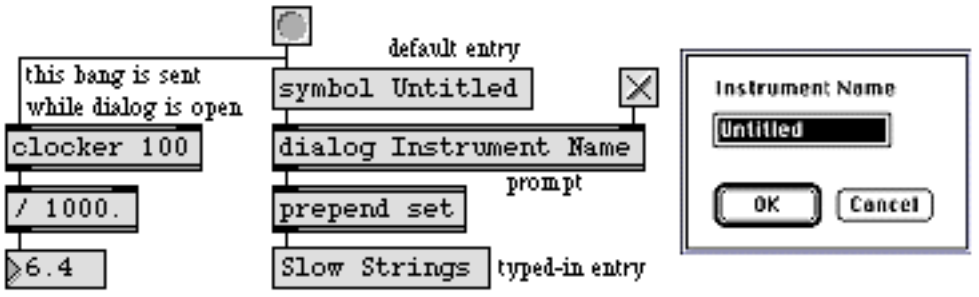
Output

symbol	If the user clicks *OK*, **dialog** makes a symbol out of the entered text (even if it's a number or it's more than one word) and sends it out its outlet with the word symbol prepended. If a nonzero number has been received in the right inlet, the typed-in message is sent out as is (without being preceded by the word symbol). This message can be displayed by prepending the word set and sending it to a **message** box (as shown in the example). If the user clicks *Cancel*, nothing is sent out.
	Since your patch continues to run while waiting for the user to type text into your dialog box, you can't count on getting the typed-in symbol immediately after sending the message that opens the dialog box.

dialog

Open a dialog box for text entry

Examples



Typed-in message is sent out when OK button is clicked; other processes continue while dialog box is open

A dialog box is opened by the **dialog** *object*

See Also

message	Send any message
opendialog	Open a dialog to ask for a file or folder
savedialog	Open a dialog to ask for a filename for saving
sprintf	Format a message of words and number

dropfile

Define a region for dragging and dropping a file

Input

(drag) When a file icon is dragged from the Finder onto a **dropfile** object in a locked patcher window, the object checks the file's type against those that it has been told to accept. If the file is of an acceptable type, the outline of the **dropfile** box is highlighted. If the mouse button is released while the cursor is inside the **dropfile** box, the **dropfile** object outputs the type and full pathname of the file out its outlets.

types The word types, followed by one or more four-letter type codes, sets the file types that will be accepted by the **dropfile** object. Example type codes for files are TEXT for text files, maxb for Max binary format patcher files, and AIFF for AIFF format audio files. types with no arguments makes the object accept all file types, which is the default setting.

border The word border, followed by a 1 or 0, sets whether the **dropfile** object draws a border around its box. The default is no border.

Arguments

None.

Output

symbol Out left outlet: When an acceptable file icon has been dragged onto **dropfile** and the mouse released within its box, the absolute pathname of the file is sent out as a single symbol. The output pathnames contain slash separators.

Absolute pathnames look like this:

“C:/Max Folder/extras/mystuff/mypatch.pat”

The **conformpath** object can be used to convert paths of one pathtype and/or pathstyle to another.

When aliases of folders are dragged onto **dropfile**, the aliases are resolved to create the output path.

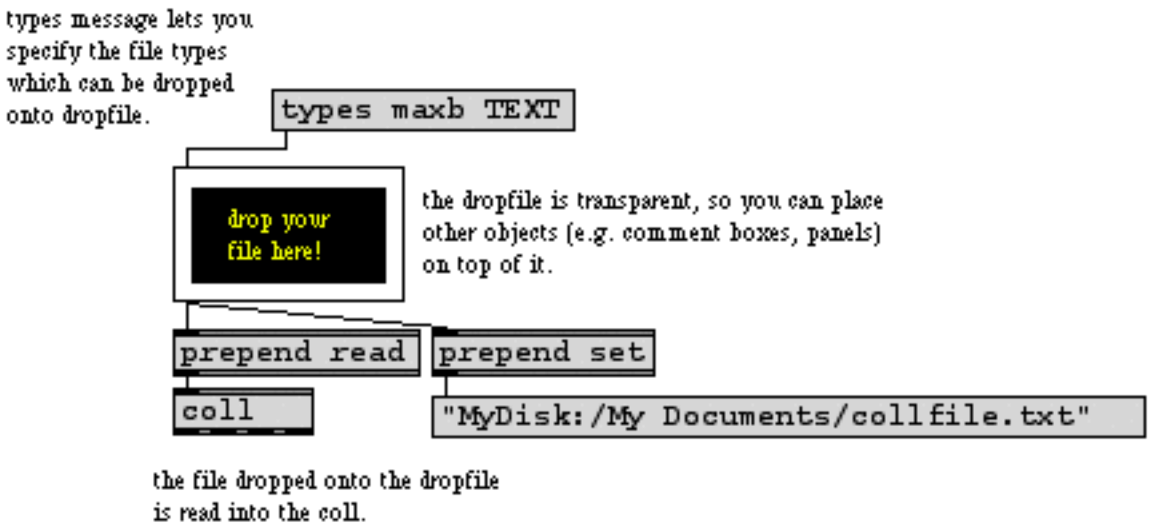
If you want to use the **dropfile** object to cause a file to be read by another object that accepts the read message with a filename argument, put a **prepend read** object between **dropfile** and the object that will open a file, as shown in the example below.

any symbol Out right outlet: The four-letter type code of the acceptable file is sent out the right outlet.

dropfile

Define a region for dragging and dropping a file

Examples


types message lets you specify the file types which can be dropped onto dropfile.

```
types maxb TEXT
prepend read
prepend set
coll
"MyDisk:/My Documents/collfile.txt"
```

the dropfile is transparent, so you can place other objects (e.g. comment boxes, panels) on top of it.

the file dropped onto the dropfile is read into the coll.

See Also

absolutepath	Convert a file name to an absolute path
relativepath	Convert an absolute to a relative path
strippath	Get filename from an absolute pathname
opendialog	Open a dialog to ask for a file or folders

drunk

Output random numbers in a moving range

Input

bang	In left inlet: Causes **drunk** to take a step of random size up or down from its currently stored value. It updates the stored value and sends it out the outlet.
int	In left inlet: The number replaces the stored value and is sent out the outlet. In middle inlet: The number is stored as the maximum value that can be output by **drunk**. (Note: If the specified maximum is less than 0 it is set to 0.) In right inlet: The number limits the step size taken in response to a bang in the left inlet. The step (up or down) will always be less than the absolute value of this number.
float	Converted to int.
list	In left inlet: The second number in the list sets the maximum value output by **drunk**, and the third number (if present) limits the step size, then the first number replaces the stored value and is sent out the outlet.
set	In left inlet: The word set, followed by a number, sets the stored value of **drunk** to that number without triggering output. The stored value is initially set in the center of the total range (1/2 the maximum value).
seed	In left inlet: The word seed, followed by a number, “seeds” the **drunk** object’s random generator, which causes a specific (reproducible) sequence of pseudo-random numbers to occur. The number 0 uses the time elapsed since system startup (an unpredictable value) as the seed, ensuring an unpredictable sequence of numbers. This unpredictable seed is used by default when the **drunk** object is created.

Arguments

int	Optional. The first argument sets an initial value for the maximum number which can be output by **drunk**. The second argument sets an initial limit on the size of random steps taken by **drunk**; the absolute value of the step size will always be less than the absolute value of this limit. If there are no typed-in arguments, the maximum value is set to 128 and the step size limit is set to 2 (movement up or down by no more than 1).

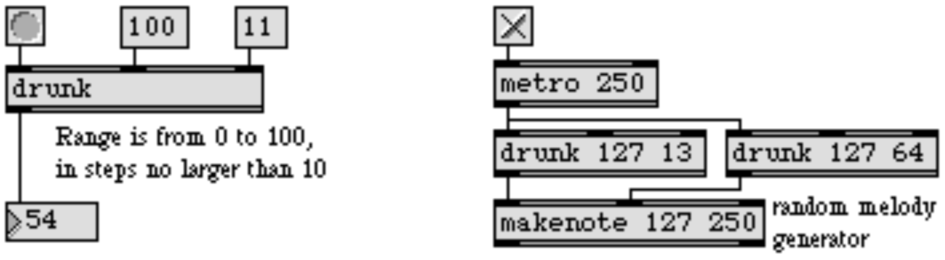
Output

int	The number sent out the outlet is automatically limited between 0 and the specified maximum value, and differs from the previously stored number by less than the maximum step size.

drunk

Output random numbers in a moving range

Examples



Numbers vary aimlessly in small steps taken within the total range

See Also

decide	Choose randomly between on and off (1 and 0)
random	Output a random number
urn	Generate random numbers without duplicates

env

Input

bang	Same as dump. Sends out a series of two-element lists, showing the array index and the value at that index for the horizontal and vertical position of each point the **env**, as specified in the object's script.
float	Converted to int.
set	The word set, followed by an array index number and a value to be stored at that index, sets the value of that array index and redraws the point, without sending anything out the outlet.
embed	The word embed, followed by any non-zero number, causes the contents of the script file to be saved as part of the patch that contains the **env** object—the next time the patch is saved—so that the **env** no longer needs to find the script file. The message embed 0 causes the **env** to forget the contents of the script file when the patch is closed. In either case, the patch must be saved after the embed message has been received in order for a change to take effect.
open	Causes the window associated with the **env** object to become visible. The window is also brought to the front. Double-clicking on the **env** object in a locked patcher has the same effect.
wclose	Closes the window associated with the **env** object.

The **env** object is a *script*-configurable user interface for function editing, oriented toward the task of editing envelope data in synthesizer patch editors.

There are two flavors of this object—**env** displays and edits the envelope in its own windows, while **envi** (pronounced "envy") is a user interface object which allows an envelope to be seen inside a patcher window. Unless otherwise noted, both objects will be referred to generically in the documentation as the **env** object.

The **env** object is configured by a script—a text file—which defines the number of points in an envelope and associates them with some number of data values. If the script is read in successfully (i.e. it contains no syntax errors), the user should be able to change displayed data points in the **env** window. **env** saves the name of the last script file read and will try to locate it the next time its owning patch is loaded.

Arguments

symbol	The **env** object takes an optional argument which is a symbol that names a script file to be read in which will define the behavior and appearance of the envelope.

Since the **envi** object is a user interface object, it doesn't have a typed-in argument. However, in both the **env** and **envi** objects, the name of the last script file read in is saved in the patcher file containing the object.

A new script file can be opened with the read message. And selecting the **envi** object and choosing Get Info… from the Object menu puts up Open Document dialog box for selecting a new script file to be read in.

Structure of an Envelope

The envelope is defined by a set of hierarchically arranged script messages. Both **env** and **envi** use identical format for script files.

Each **env** object consists of a window (technically in **envi**, a box in a patcher window), a number of groups, each of which contain points which are logically connected. Each point contains horizontal and/or vertical aspects, and each aspect can contain one or more display scales, which map internal data values to those displayed on the legend of the envelope window.

Script Messages

The format of a script file consists of #E followed by a message keyword (such as group or point), followed by that message's arguments. See the Script Examples section below for examples.

The window message

Defines parameters applying to the entire **env** object and its display.

symbol	1. Title of the envelope window (doesn't apply to **envi**). To use spaces in the title, use single "smart" quotes (option-right bracket and option-right brace).
int	2. Horizontal size. Size of the window (or box, in the case of **envi**) in pixels. For the window, the size will be actually be 15 pixels larger to accommodate the scroll bars.
int	3. Vertical size.
int	4. Number of groups. Each group will be defined in subsequent group messages (see below).
int	5. Number of data values that define the envelope(s).
int	6. Left margin. Distance in pixels from left edge of the window (box) where the envelope and text legend is drawn.
int	7. Bottom margin. Distance in pixels from bottom edge of the window (box) where the envelope is drawn.
int	8. Top margin. Distance from the top of the window (box) where the envelope is drawn. This should take into account the legend (which is 15 pixels), so a value of 20 or more pixels is suggested.

The group message

Defines a group of logically connected points, what would usually be thought of as an "envelope"—but the **env** object allows an arbitrary number of groups in a single window.

int	1. Group number. Specifies the group (starting at 1) being defined.
symbol	2. Group name. Precedes the name of any specific parameter and value in an envelope legend display. The word none can be used to indicate that no group name is desired.
int	3. Number of points in this group. Each will be defined below with a point message.
int	4. Visible. 1 if this group is initially visible, 0 if it isn't.
int	5. Display flags. 1 if you only want the parameter names and values of a point being dragged. 0 if you want all the parameter names and values displayed when a point in the group is being dragged. Other display flags may be defined later.
int	6. (Optional) Color. 1-15 as an index into the color palette and correspond to the colors set in the Edit Colors... patch accessed via the Options menu.

The point message

Defines the appearance of a "point" in an envelope.

int	1. Point number being defined. The first point in any group is number 1.
int	2. Button size (in pixels) of the round or square "button" centered at this point.
int	3. Button flags. The rightmost bit (i.e. 0 or 1) is 0 if the button is to be square and 1 if the button is to be round. Bit 1 (i.e. 0 or 2) is 1 if the button is solid, 0 if it is transparent. Bits 2-6 (inclusive) specify an index for a black and white pattern. Use ResEdit to examine the System File and look at PAT# ID 1 for the indices of common black and white patterns.
int	4. Line-from point. If non-zero, specifies another point, which should always be numbered less than this point, which is to be connected to this point with a line. This connection is only a display property. Logical dependencies between points are specified in the horiz and vert messages below.

The horiz and vert messages

These messages define the two directional aspects of each point. Most of the "meat" of the envelope specification is contained in these messages. If you wish to keep one of the directions fixed, you need not define that direction for a particular point. The arguments to horiz and vert are identical, except where noted.

symbol	1. Parameter name. The name (e.g. 'Rate 1') associated with moving the point in this direction. none can be used if there is no parameter name associated with this point.
int	2. Data index. The index into the array of data values (starting at 0) corresponding to the value of this parameter. If there is no data associated with this direction, use -1 (this will not be uncommon for one or more directions of one or more points in an envelope). When a list containing this data index and a value is sent to the **env** object, this point will move accordingly. Note that all data values are stored as integers. You can display a floating point number in the legend for this parameter by defining a scale expression or table (see the scale message below).
int	3. Minimum value of this parameter.
int	4. Maximum value of this parameter.
int	5. Initial value of this parameter.
int	6. Increment of this parameter. Not currently supported, should be set to 1.
symbol	7. Unit name. The units of this parameter (e.g. ms for milliseconds or % for percentage). none may be used if the units are not tied to any particular units, such as the rate and level units on Yamaha synthesizers).

When two points are "tied together" in the horizontal or vertical direction it means that changes in one point are linked to others. Ties are expressed in terms of higher numbered points being tied to lower numbered ones. There are two types of ties—absolute and relative. An absolute tie means that a point changes its position on the screen to assume the exact value of another point. A relative tie, which is very common for horizontal aspects, means that the location of any point on the screen is based on a distance from another point. The common envelope shown in the second Script Example section below has point 2 with a relative horizontal tie to point 1, point 3 with a relative horizontal tie to point 2 (and hence to point 1), and point 4 with a relative horizontal tie to point 3. If point 1 is allowed to move left and right (as for example if there were an initial delay for the envelope, all the other points would move as well. None of the points are vertically tied to each other, although in a DX7 envelope which has a non-zero final level, it is customary to tie points, points 1 and 4 would be absolutely vertically tied. You cannot tie the horizontal direction of one point to the vertical direction of another.

int	8. Absolute tie point. Point number that this point is absolutely tied to (must be less than this point number). This point will appear at the exact same horizontal or vertical position as the point it is tied to. Use 0 if this point is not tied.

int	9. Fixed. If this point is fixed at a particular position on the screen, use 1. Otherwise use 0. This may be true for the horizontal or vertical direction of the first (leftmost) point in an envelope.
int	10. Relative tie point. Point number that this point is relatively tied to (must be less than this point number) in this direction. This point's position will be an offset (depending on its value) from the position of the point being tied to in the horizontal or vertical direction. Use 0 if this point is not relatively tied to other points in this direction (commonly true for the vertical direction).
int	11. Positive direction. Sets which direction the value of a point increases. For the vertical direction, 0 indicates that the value increases as the cursor is moved to the top of the screen, while 1 indicates that the value increases as the mouse is moved to the bottom of the screen. For the horizontal direction, 0 indicates that the value increases as the cursor is moved to the right, while 1 indicates that the value increases as the cursor is moved to the left.
int	12. Coverage size. Determines how many pixels the range of the parameter is mapped into. For a garden variety envelope, you generally use most of the entire vertical space for the vertical direction, so you would use a formula like: <window vertical size> - <legend height> - <top margin> - <bottom margin> For the horizontal direction, the amount of space you use should be determined by the number of points in the envelope, and how much scrolling you want to require the user to do if the envelope is stretched to its maximum width.

The scale message

Defines a conversion between the internal values (integers) used to store the data in an envelope and their displayed values, which may be floating point numbers. When envelope parameters represent physical quantities, manufacturers often use scale factors. In the scale message, you can specify a mathematical expression to convert the internal format to another integer or floating point number which is displayed in the legend.

A scale can be expression in the form of the arguments to the **expr** object, or it can be a list of values (including symbols) to which the internal data values map.

Each direction can have an arbitrary number of scales, each of which is applicable over a specified range. If there is no scale which applies to a data value, the legend will display the internal data value. One use of a scale in this context might be if the lowest value of an envelope signified "Off"—you could have a scale that mapped 0 to the word "Off" but left the other values unchanged.

int	1. Minimum. Lowest value for which this scale applies.
int	2. Maximum. Highest value for which this scale applies.
int	3. Floating-point digits. Number of digits after the decimal point used to display floating-point numbers in the legend.

symbol 4. The word is or table. Determines whether what follows is interpreted as a mathematical expression or a table of values used for mapping.

5. Additional data. For expressions: $i1 represents the internal data being mapped to the legend. Examples:

is $i1 * .07;Multiplies the internal value by a scale factor

is $i1 - 1;Subtracts 1 from the internal value

is ($i1-1)*.07;Compound expression

is 100 - $i1;Inverting an internal value

For tables: a list of values which map successive values of the internal data separated by spaces. The table can contain up to 240 elements. Use additional scale messages for larger tables. Example:

table Off 10 20 30 40;

Here, the minimum value will be mapped to the word "Off ", next value to 10, next value to 20 etc.

Other Example scale messages:

#E scale 0 0 0 table Off; (Maps the minimum value to the word "Off.")

#E scale 1 10 2 is $i1 * .04;(Scales additional values by .04 and prints as floating-point number with 2 decimal places.)

The phase message

This message specifies that the previously defined vert aspect of a point has a signed component. Either the parameter of the envelope can be a negative number, or there is a separate data value that represents the phase (0 for negative, 1 for positive). The phase message must immediately follow the vert message it modifies.

The comment message

This message begins a comment in the envelope script, which must be contained on a single line and terminated with a semicolon.

The end message

This message is required at the end of an envelope script. It reconfigures the **env** object and changes the display in its window or box if necessary. It has no arguments.

Script Examples

The following script defines an envelope which consists of 4 groups of individual points which are used in an early reflection tap editor. The horizontal position of the point determines a delay and the vertical position determines a percentage of the original signal to repeat. A picture is shown after the script.

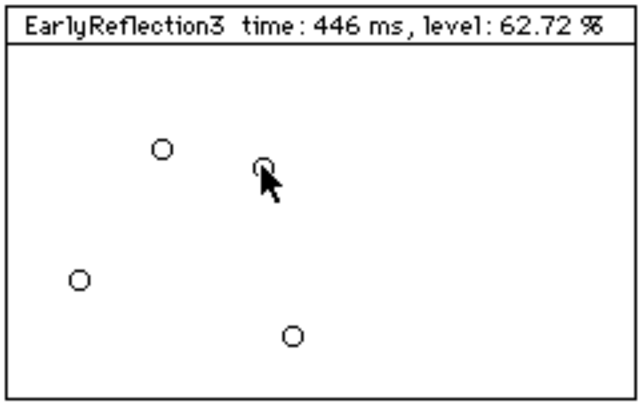
```
#E window ERFEnv 400 148 4 96 8 8 24;
#E group 1 EarlyReflection1 1 1 1;
#E point 1 8 1 0;
#E horiz time 0 1 500 1 1 ms 0 0 0 0 100;
#E vert level 1 0 1024 0 1 % 0 0 0 0 100;
#E scale 0 1024 2 is $i1 * .0977;

#E group 2 EarlyReflection2 1 1 1;
#E point 1 8 1 0;
#E horiz time 2 1 500 1 1 ms 0 0 0 0 100;
#E vert level 3 0 1024 0 1 % 0 0 0 0 100;
#E scale 0 1024 2 is $i1 * .0977;

#E group 3 EarlyReflection3 1 1 1;
#E point 1 8 1 0;
#E horiz time 4 1 500 1 1 ms 0 0 0 0 100;
#E vert level 5 0 1024 0 1 % 0 0 0 0 100;
#E scale 0 1024 2 is $i1 * .0977;

#E group 4 EarlyReflection4 1 1 1;
#E point 1 8 1 0;
#E horiz time 6 1 500 1 1 ms 0 0 0 0 100;
#E vert level 7 0 1024 0 1 % 0 0 0 0 100;
#E scale 0 1024 2 is $i1 * .0977;

#E end;
```


EarlyReflection3 time: 446 ms, level: 62.72 %

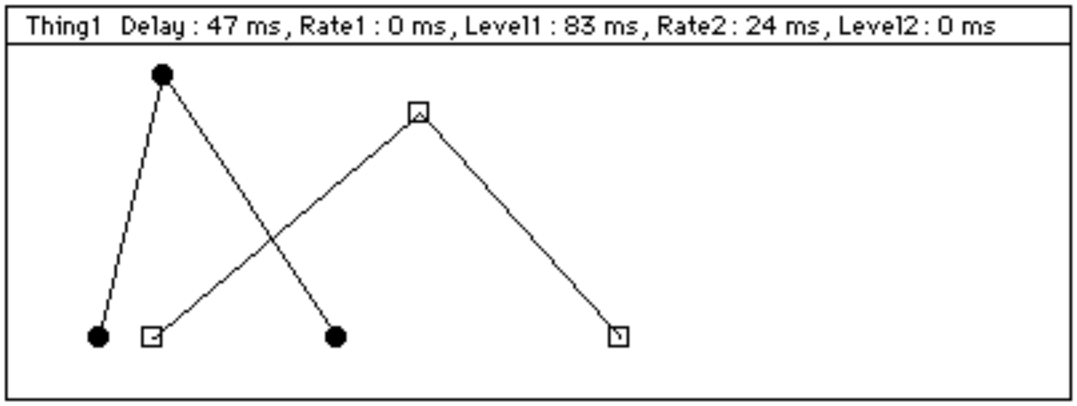
Picture of object for Script Example #1

The following script defines a two groups with more traditional synthesizer amplitude envelopes that have three points. The first point is fixed in the vertical direction but moves horizontally. The other two points move in both directions, and all three points are connected by a line. A picture is shown after the script.

```
#E window TestEnv 400 148 2 10 8 8 24;

#E group 1 Thing1 3 1 0;
#E point 1 8 0 0;
#E horiz Delay 0 0 99 0 1 ms 0 0 0 0 100;
#E vert none -1 0 99 0 0 none 1 0 0 0 100;
#E point 2 8 0 1;
#E horiz Rate1 1 0 99 50 1 ms 0 0 1 1 100;
#E vert Level1 2 0 99 50 1 ms 0 0 0 0 100;
#E point 3 8 0 2;
#E horiz Rate2 3 0 99 50 1 ms 0 0 2 1 100;
#E vert Level2 4 0 99 50 1 ms 0 0 0 0 100;

#E group 2 Thing2 3 1 0;
#E point 1 8 3 0;
#E horiz Delay 5 0 99 0 1 ms 0 0 0 0 100;
#E vert none -1 0 99 0 0 none 1 0 0 0 100;
#E point 2 8 3 1;
#E horiz Rate1 6 0 99 50 1 ms 0 0 1 1 100;
#E vert Level1 7 0 99 50 1 ms 0 0 0 0 100;
#E point 3 8 3 2;
#E horiz Rate2 8 0 99 50 1 ms 0 0 2 1 100;
#E vert Level2 9 0 99 50 1 ms 0 0 0 0 100;
#E end;
```


Thing1 Delay : 47 ms, Rate1 : 0 ms, Level1 : 83 ms, Rate2 : 24 ms, Level2 : 0 ms

Picture of Object for Script Example #2

Input Messages

Because it can have an arbitrary number of data values, the **env** object has only one inlet. The envelope data is stored in an array. The script file specifies how array indices correspond with horizontal and vertical aspects of the points in an envelope.

list A list received by **env** stores a new value in a data point. The first number in the list specifies the location (array index), and the second number is the data value to store at the location. The **env** object limits the range of its input values, according to the minimum and maximum of each data point specified in the script file.

The **funnel** object takes a number in one of its inlets and outputs a list with the first element being the index of the inlet and the second element being the incoming number. It was designed to be used to prepare the lists required by the **env** object.

int If the number is between 0 and the maximum array index, **env** outputs a list containing the index followed by the data value at the array index.

show The word **show**, followed by a group number, makes that group visible. Followed by two numbers, makes a range of groups visible from the first to the second number.

hide The word **hide**, followed by a group number, makes that group invisible. Followed by two numbers, makes a range of groups invisible from the first to the second number.

open Opens the **env** object's display window if its closed, or brings it to the front. Doesn't apply to the **envi** object.

read Puts up a standard Open Document dialog for the user to select a new script file for configuring the object.

dump Outputs all the current data values of the envelope, as successive two element lists. The first number is the data index and the second is the data value.

Output

list When the mouse button is released or a number is received in its inlet, **env** sends lists outs its outlet which consist of two numbers. The first is an array index and the second is the new value at that index. Only newly modified values are output. When **env** receives the dump message in its inlet, all data values are sent out in this list format.

The **spray** object takes a list as input and sends the second element out the outlet number specified by the first element. It was designed to distribute the lists output by the **env** object to individual outlets for display by number boxes or to send to librarian editor objects such as **libto**.

Using an Envelope Window or Box

The envelope display has two areas separated by a horizontal line—the upper area of 15 pixels contains a legend of text in 9 point Geneva that indicates the names and values of the points the user is currently changing. The lower area contains the actual groups of points which may or may not be connected by lines.

The use of the **env** object's window (or the **envi** object's box) is simple—just click on one of the visible points. With no modifier keys held down, data values are incremented by a pixel's worth of movement. How much this amounts to is determined by the ratio of each direction's Coverage size argument to its parameter range (difference between maximum and minimum values). For example, in the first example script above, there are 1024 data points and a Coverage size of 100, so moving the cursor one pixel changes the value by 1024/100, or about 10.

With the Shift key down, movement of a point being dragged is constrained to the direction the cursor moves in first. Releasing the Shift key at any time removes the constraint.

With the Command key on Macintosh or Control key on Windows held down, mouse movement is in "fine mode"—no matter what the ratio of parameter range to Coverage size, the parameter data is changed by 1 with each pixel you move the mouse.

Fine mode can be entered or left instantaneously by pressing or releasing the Command key on Macintosh or Control key on Windows while dragging the mouse.

See Also

envi	Script-configurable envelope in a patcher window
funbuff	Store x,y pairs of numbers together
funnel	Tag data with a number that identifies its inlet
line	Output numbers in a ramp from one value to another
multislider	Multiple slider and scrolling display
spray	Distribute an integer to a numbered outlet

envi

Script-configurable envelope in a patcher window

The **envi** object is the patcher window version of the **env** object. The discussion of the **env** object covers both objects.

error

Input

int	The **error** object allows you to catch errors and output them as Max messages. A non-zero number starts the error object "listening" for Max errors. The **error** object must be listening to produce any output. A 0 turns off listening.
float	Converted to int.

Arguments

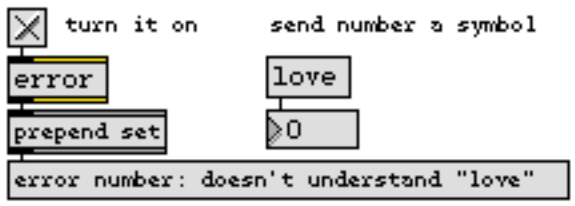
None.

Output

symbol — Any Max error generated by any object in any patch while the **error** object is listening is sent out the outlet preceded by the symbol error. The messages are output as individual words so you can check for specific failures.

If you want to strip off the initial error message from the object's output, use a **route error** object. If you want to use the **error** object's output as a message, put a **prepend read** object between **route error** and the object that will process the error message.

Examples



Intercept error messages

See Also

print	Print any message in the Max window

expr

Evaluate a mathematical expression

Input

int	The number received in each inlet will be stored in place of the $i or $f argument associated with it. (Example: The number in the second inlet from the left will be stored in place of the $i2 and $f2 arguments, wherever they appear.)
float	The number in each inlet will be stored in place of the $f or $i argument associated with it. The number will be truncated by a $i argument.
symbol	The word symbol, followed by the name of a **table**, will be stored in place of the $s argument associated with that inlet, for accessing values stored in the **table**.
bang	In left inlet: Evaluates the expression using the values currently stored.
list	In left inlet: The items of the list are treated as if each had come in a different inlet, and the expression is evaluated. If the list contains fewer items than there are inlets, the most recently received value in each remaining inlet is used.

Any of the above messages in the left inlet will evaluate the expression and send out the result. If a value has never been received for each changeable argument, that value is considered 0 when the expression is evaluated.

The number of inlets is determined by how many changeable arguments are typed in. The maximum number of inlets is 9.

set	In left inlet: The word set, followed by one or more numbers, treats those numbers as if each had come in a different inlet, replacing the stored value with the new value, but the expression is not evaluated and nothing is sent out the outlet. If there are fewer numbers in the message than there are inlets, the stored value in each remaining inlet stays unchanged.

Arguments

Obligatory. The argument is a mathematical expression, in a format resembling the C programming language. The expression is made up of numbers, arithmetic operators such as **+** or *****, comparisons such as **<** or **>**, C functions such as min() or pow(), names of **table** objects, and changeable arguments ($i, $f, and $s) for ints, floats, and symbols received in the inlets.

int or float	Numbers can be used as constants in the mathematical expression.
$i or $f	A changeable int argument is specified by $i or $f and an inlet number (example: $i2). The argument will be replaced by numbers received in the specified inlet.
$s	The argument $s and an inlet number is replaced by the name of a **table** to be accessed. The argument should be immediately followed by a number in brackets specifying an address in the **table**. (Examples: $s2[7] or $s3[$i1].)

(other) Arithmetic operators understood by **expr** are: +, -, *, /, %. Other operators are ~ (one's complement), ^ (bitwise exclusive or), &, &&, |, ||, and ! (not).

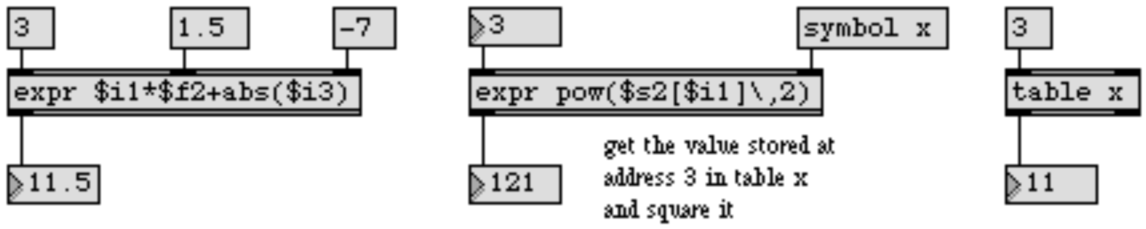
Many C language math functions can be understood by **expr**. A function must be followed immediately by parentheses containing any arguments necessary to the function. If the function requires a comma between arguments, the comma must be preceded by a backslash (\) so that Max will not be confused by it. For example: pow($i1\,2).

C language functions understood by **expr** are: abs, min, max, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, int (convert to integer), float (convert to float), pow, sqrt, fact (factorial), exp (power of e to x), log10 (log), ln or log (natural log), and random. Additional functions can be added by means of external code resources placed in Max's startup folder.

Output

int or float	The output is the result of the evaluated expression.

Examples



Combine many calculations into one object, even using functions not available in other objects

See Also

if	Conditional statement in if/then/else form
vexpr	Evaluate a math expression for a list of different inputs
Tutorial 38	**expr** and **if**

filedate

Report the modification date of a file

Input

symbol	A file pathname as a symbol. An absolute pathname looks like this: 'MyDisk:/Max Folder/extras/filename'

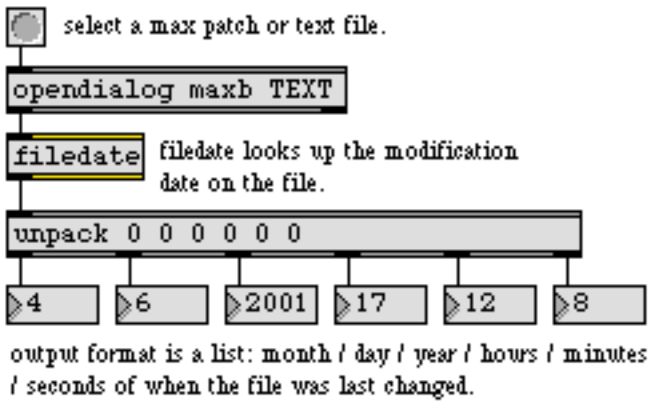
Arguments

None.

Output

list	Sends the date that the file was last changed as a list (month, day, year, hours, minutes and seconds).

Examples


select a max patch or text file.
opendialog maxb TEXT
filedate
filedate looks up the modification date on the file.
unpack 0 0 0 0 0 0
4
6
2001
17
12
8
output format is a list: month / day / year / hours / minutes / seconds of when the file was last changed.

filedate *displays how recently a file has been changed*

See Also

date	Report current date and time
filein	Read in a file of binary data
filepath	Report information about the current search path
folder	List the files in a specific folder
opendialog	Open a dialog to ask for a file or folder

filein

Read in a file of binary data

Input

int	Specifies a byte offset in a binary file, and outputs the data stored at that point in the file. In left inlet: The byte contained at that offset in the file is sent out the left outlet. In middle inlet: The 16-bit word contained at that byte offset in the file is sent out the left outlet as an unsigned (short) integer. In right inlet: The 32-bit word contained at that byte offset within the file is sent out the left outlet as an unsigned (long) integer.
list	In left inlet: The second number in the list is received in the middle inlet, then the third number in the list (if present) is received in the right inlet, and then the first number in the list is received in the left inlet. Output is sent out the left outlet in the corresponding order.
read	Displays a standard file dialog to select a file to be read into memory. If the word read is followed by a filename found in Max's search path, that file will be automatically read into memory.
spool	Displays a standard file dialog to select a file, which will be accessed from disk whenever an int is received. If the word spool is followed by a filename found in Max's search path, that file will be automatically pointed to for future access. This method of accessing a file occupies less RAM, but does not output data immediately at interrupt level in response to an int message.
fclose	Closes the file being read, making **filein** no longer respond to int or list messages.

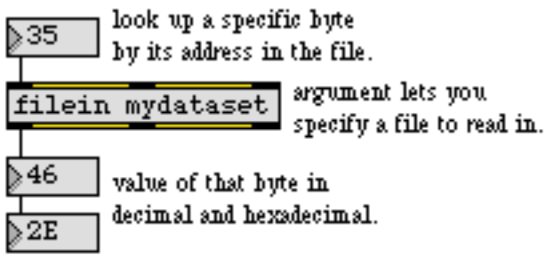
Arguments

symbol	Optional. Specifies a filename to be read into the **filein** object automatically when the patch is loaded. If the filename is followed by a second argument, spool, the file will be accessed from disk rather than read into memory.

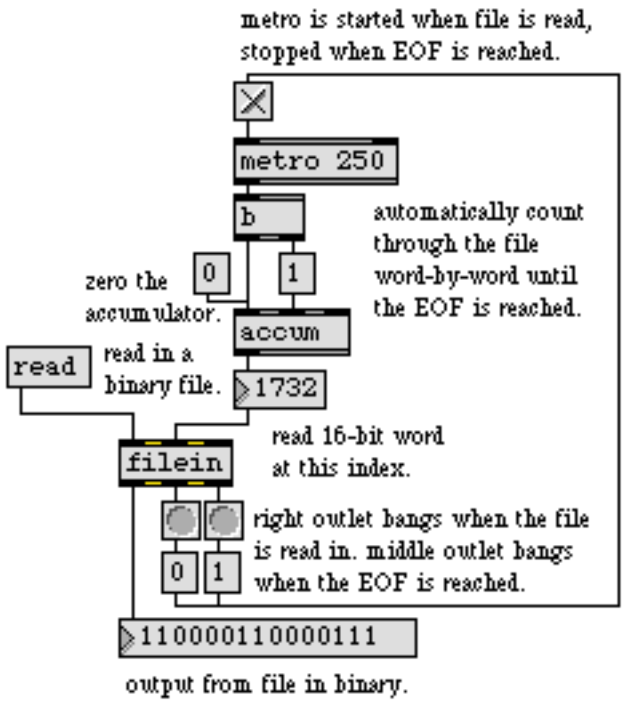
Output

int	Out left outlet: An unsigned integer representing the 8, 16, or 32 bits stored in the file at the location specified by the input int.
bang	Out middle outlet: When a number greater than or equal to the number of bytes in the file is received in an inlet, a bang is sent out signifying that the end of the file (EOF) has been reached. Out right outlet: Signifies that a read or spool operation has been completed. This bang indicates that the file has been accessed successfully and that **filein** is ready to receive int messages.

Examples


35
look up a specific byte by its address in the file.
filein mydataset
argument lets you specify a file to read in.
46
2E
value of that byte in decimal and hexadecimal.

Retrieve data from any binary file


metro is started when file is read, stopped when EOF is reached.
metro 250
b
automatically count through the file word-by-word until the EOF is reached.
zero the accumulator.
0
1
accum
read
read in a binary file.
1732
read 16-bit word at this index.
filein
right outlet bangs when the file is read in. middle outlet bangs when the EOF is reached.
0
1
110000110000111
output from file in binary.

Output the content of a file in 8-, 16-, or 32-bit chunks

See Also

text	Format messages as a text file

filepath

Report information about the current Max search path

Input

any symbol	The pathname of a file in the search path as a symbol. Input pathnames can contain slashes, colons, or backslashes as separators. A pathname looks like this: "drive:/folder/filename.ext" (absolute pathname) "./mypatches/steaksauce.ext" (relative pathname)
bang	A bang causes the currently saved path name(s) to be output as a list.
append	The word append, followed by a symbol which specifies a folder, adds the folder to the list of paths (but does not save it in the Preferences file).
set	The word set, followed by the name of a Max search path type (search, startup, help, action, or default), sets the current search path to the type specified.
revert	Causes the pathnames to be reset to the last set of Max file preferences to be saved.
clear	Causes the currently specified search path to be cleared.

Arguments

symbol	Obligatory. Specifies one of the Max search path types (search, startup, help, action, or default)
int	Optional. A number greater than zero specifies a slot in the Preferences file. If the argument is 0 or no number is supplied, the path will not be saved in the Preferences file—you can use this feature to create temporary search paths for a patch. The action, help, and startup paths only have one slot. The search path can have up to 256 slots (normally there are about 8). The default path is never saved in the Preferences file.

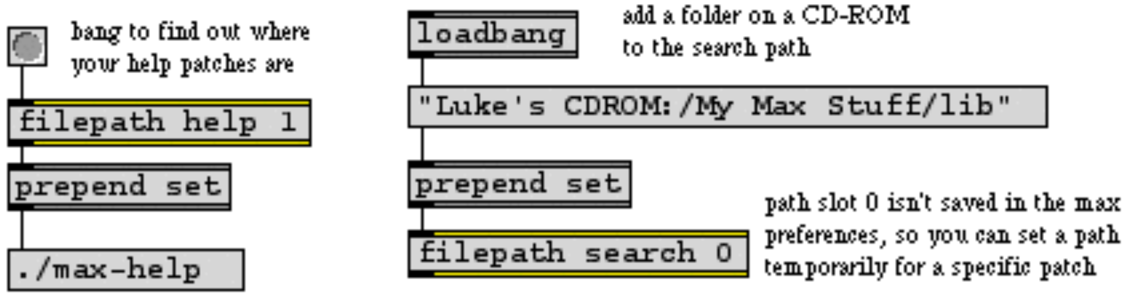
Output

symbol	The currently stored path name in response to a bang.

filepath

Report information about the current Max search path

Examples


bang to find out where your help patches are
filepath help 1
prepend set
./max-help
loadbang
add a folder on a CD-ROM to the search path
"Luke's CDROM:/My Max Stuff/lib"
prepend set
filepath search 0
path slot 0 isn't saved in the max preferences, so you can set a path temporarily for a specific patch

Use **filepath** *to check your search path or temporarily set search path slots for a patch*

See Also

conformpath	Convert paths of one pathtype and/or pathstyle to another
filedate	Report the modification date of a file
filepath	Report information about the current search path
folder	List the files in a specific folder
opendialog	Open a dialog to ask for a file or folder

float / f

Store a decimal number

Input

float	In left inlet: The number replaces the currently stored value and is sent out the outlet. In right inlet: The number replaces the stored value without triggering output.
bang	In left inlet: Sends the stored value out the outlet.
set	In left inlet: The word set, followed by a number, replaces the stored value without triggering output.
send	In left inlet: The word send, followed by a name of a **receive** object, sends the number stored in the **float** object to all **receive** objects with that name, without sending it out the **float** object's outlet.
int	Converted to float.

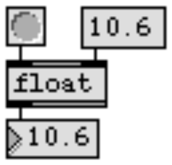
Arguments

float	Optional. Sets an initial value to be stored in **float**. If there is no argument, the initial value is 0.0. A float argument by itself, without the word float, is another way of creating and initializing a **float** object.

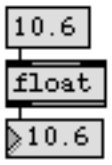
Output

float	A number is stored in **float** as a single-precision floating point number. The precision possible in the decimal portion of the number decreases as the integer part increases. Note: Because of the way decimal numbers are stored, a float value saved in a patcher file might be slightly altered when the file is reopened.

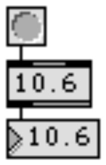
Examples



Output the stored value



Replace stored value and output it



Initial value is given

See Also

int	Store an integer value
pv	Share variables specific to a patch and its subpatches
value	Share a stored message with other objects
Tutorial 21	Storing numbers
Data Structures	Ways of storing data in Max

flush

Provide note-offs for held notes

Input

int In left inlet: The number is treated as the pitch value of a pitch-velocity pair and the note is sent out.

In right inlet: The number is stored as the velocity to be paired with numbers received in the left inlet.

list In left inlet: The numbers must be ints. The first number is treated as the pitch, and the second number is treated as the velocity, of a pitch-velocity pair, and the numbers are sent out the outlets.

bang In left inlet: Immediately sends note-offs for any pitches that have passed through as note-ons but not as note-offs by sending 0 out its right outlet followed by a pitch value out its left outlet.

clear In left inlet: Erases any numbers held by **flush**, without sending any note-offs.

Arguments

None.

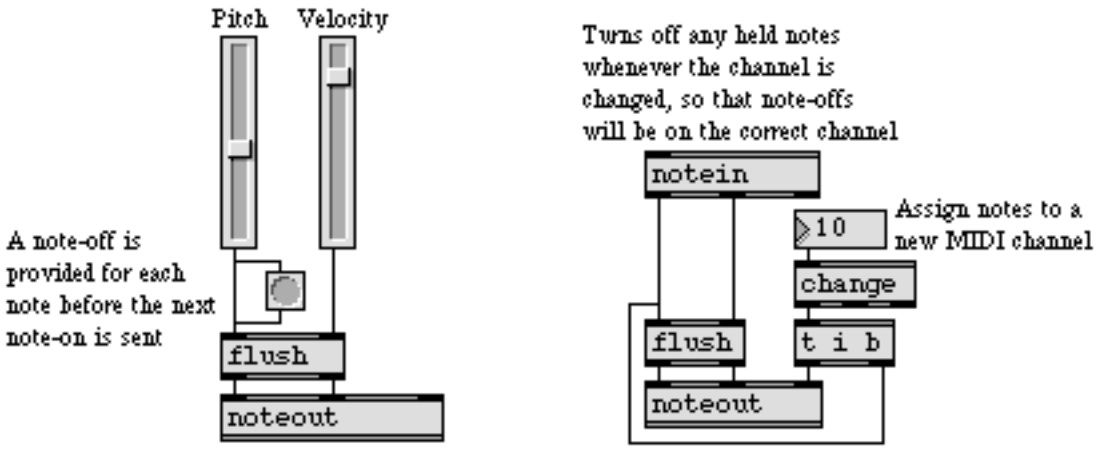
Output

int Out left outlet: The output is the pitch of the note-on or note-off.

Out right outlet: The number is the velocity of the note-on or note-off.

The **flush** object keeps track of the notes that have passed through it. When a bang is received in the inlet, note-off messages are provided for any notes that have passed through as note-ons only.

Examples



Pitch
Velocity
A note-off is provided for each note before the next note-on is sent
flush
noteout
Turns off any held notes whenever the channel is changed, so that note-offs will be on the correct channel
notein
10
Assign notes to a new MIDI channel
change
flush
t i b
noteout

Make sure all notes are turned off by providing note-offs for held notes

See Also

bag	Store a collection of numbers
borax	Report current information about note-ons and note-offs
makenote	Generate a note-off message following each note-on
midiflush	Send note-offs for hanging note-ons in raw MIDI data
offer	Store x,y pairs of numbers temporarily
stripnote	Filter out note-off messages, pass only note-on messages
sustain	Hold note-off messages, output them on command
Tutorial 13	Managing note data

folder

List the files in a specific folder

Input

bang Gets the names of all files of a specific type within a specific folder, and outputs those names to be placed in a **message** object or a pop-up **umenu** object.

symbol Specifies the pathname of a folder in the search path, and causes the contents of that folder to be output for storage in a **umenu** or a **message**. Input pathnames can contain slashes, colons, or backslashes as separators.

A pathname looks like this:

"drive:/folder/filename.ext" (absolute pathname)
"./mypatches/steaksauce.ext" (relative pathname)

If the pathname contains any spaces, you will need to enclose the pathname in double quotes in order to cause **folder** to understand the pathname as a single argument. Alternatively, you can precede each space with a backslash (\) so that **folder** won't treat that space as a special character.

types The word types, followed by one or more four-letter type codes, sets the file types that the **folder** object will look for in the specified folder. Example four-letter type codes for files are TEXT for text files, maxb for Max binary format patcher files, and AIFF for AIFF format audio files.

By default, the **folder** object looks for TEXT and maxb (Max binary) files.

int Same as bang.

Arguments

symbol Optional. Specifies the absolute path to a folder on any mounted volume.

Output

clear Out left outlet: When a pathname or a bang is received in the inlet, the first message that is sent out the left outlet is clear, which is intended to erase the contents of a receiving **message** or **umenu** object.

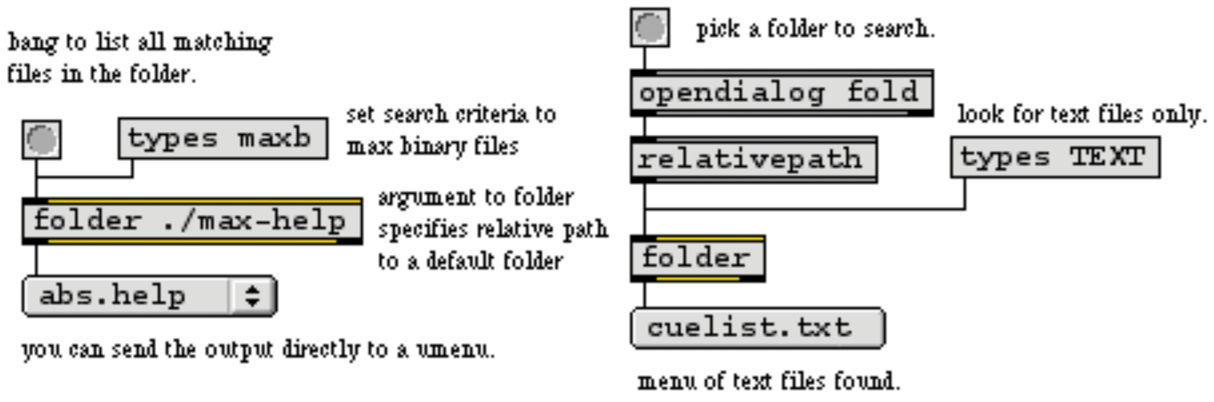
append Out left outlet: Immediately following the clear message, each filename in the specified folder is sent out in alphabetical order preceded by the word append.

int Out right outlet: When a pathname or a bang is received in the inlet, the number of items in the folder is sent out the right outlet.

folder

List the files in a specific folder

Examples


bang to list all matching files in the folder.
types maxb
set search criteria to max binary files
folder ./max-help
argument to folder specifies relative path to a default folder
abs.help
you can send the output directly to a umenu.
pick a folder to search.
opendialog fold
relativepath
look for text files only.
types TEXT
folder
cuelist.txt
menu of text files found.

Read in filenames from a folder, then call them up from a pop-up menu

See Also

conformpath	Convert paths of one pathtype and/or pathstyle to another
filein	Read in a file of binary data
filepath	Report information about the current search path
opendialog	Open a dialog to ask for a file or folder
pcontrol	Open and close subwindows within a patcher

follow

Compare a live performance to a recorded performance

Input

record — Starts recording integers received in the inlet.

bang — Starts playing back the sequence stored in **follow**.

start — The word start by itself has the same effect as bang. The word start, followed by a number, plays the stored sequence at a tempo determined by the number. The message start 1024 indicates normal tempo. If the number is 512, **follow** plays the sequence at half the original recorded speed, start 2048 plays it back at twice the original speed, and so on.

follow — The follow message is the main feature that distinguishes **follow** from **seq**. In effect, **follow** is like a score reader, comparing a live performance with the one previously stored.

The word follow, and a number, causes **follow** to begin comparing incoming numbers to its own stored numbers, beginning at the specified index (the specified event in its own stored sequence). When **follow** is following, and a number is received that matches the number recorded in **follow**, it sends out the index of that number.

The **follow** object is a forgiving score reader, and will try to follow along even if the incoming numbers do not exactly match the recorded sequence. If a number arrives that does not match the next number, or either of the two subsequent numbers in the sequence, **follow** does nothing. If a number arrives that matches a number up to two notes ahead in the sequence, **follow** assumes that the performer simply missed a note or two, and jumps ahead to the matched number.

stop — Stops **follow** from recording, playing, or following. A stop message need not be received before switching directly from recording to playing, following to recording, etc.

next — Causes **follow** to send out the index and the stored number it is currently trying to match, and move on to the next number.

append — Starts recording at the end of the stored sequence, without erasing the existing sequence.

int — When **follow** is recording, the numbers received in its inlet are recorded as a sequence. The numbers may be bytes of MIDI messages (from **midiformat** or **midiin**), exactly as with the **seq** object. However, **follow** differs from **seq** in its ability to record individual integers; with **follow** you can record notes as a single pitch value. Whether the performance is recorded as complete MIDI messages or just as note-on pitches, **follow** can effectively step through the note-on pitch numbers later, when following a performance.

When **follow** is following, numbers received in its inlet are compared to the numbers recorded in the sequence. When a number is received that matches the number in the sequence, **follow** sends out the index of that number.

float Converted to int.

delay The word delay, followed by a number, sets the onset time, in milliseconds, of the first event in the recorded sequence.

hook The word hook, followed by a float, multiplies all the event times in the stored sequence by that number. For example, if the number is 2.0, all event times will be doubled, and the sequence will play back twice as slowly. Multiplications can even be performed while the sequence is playing.

write Opens a standard Save As dialog box to save the **follow** sequence as a file.

read The word read with no arguments puts up a standard Open Document dialog box for choosing a sequence file to load into **follow**. If read is followed by a symbol filename argument, the named file is located and loaded into **follow**.

print Prints the first few events of the recorded sequence in the Max window.

dump Calls up the standard Open Document dialog box, so that a previously recorded sequence or standard MIDI file can be opened as text and displayed in a new Untitled text window. This in fact has no direct effect on the **follow** object, but does allow you to view or edit a sequence, save your changes in a file, then load the new file into **follow** with a read message.

Arguments

any symbol Optional. The argument is the name of a file containing a previously recorded sequence, to be read into **follow** automatically when the patch is loaded.

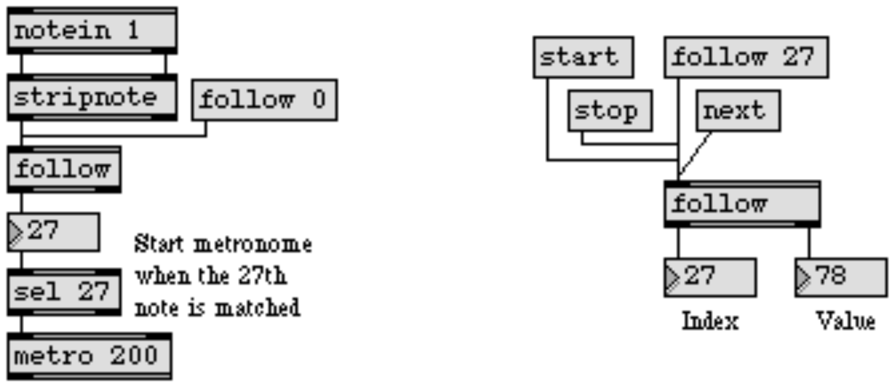
Output

int Out left outlet: When **follow** is following, and the number received in the inlet matches the next number in the stored sequence (or one of the two numbers after that), the index of the matched number is sent out. The index of the next number is also sent out when a next message is received.

Out right outlet: When **follow** receives a bang or a start message, the recorded numbers are played back. When **follow** is following, and a next message is received, the next number in the recorded sequence is sent out.

Compare a live performance to a recorded performance

Examples



A note that matches the recorded note can trigger a process, or the notes can be stepped through

See Also

seq	Sequencer for recording and playing MIDI
detonate	Graphic score of note events
Tutorial 35	**seq** and **follow**
Sequencing	Recording and playing back MIDI performances

forward

Send remote messages to a variety of objects

Input

anything	Sends any message to all **receive** objects which share the name currently referred to by **forward**.
send	The word send, followed by the name of a **receive** object, sets the destination for any subsequent messages received by the **forward** object. This ability to change the destination of messages on the fly distinguishes **forward** from the **send** object.

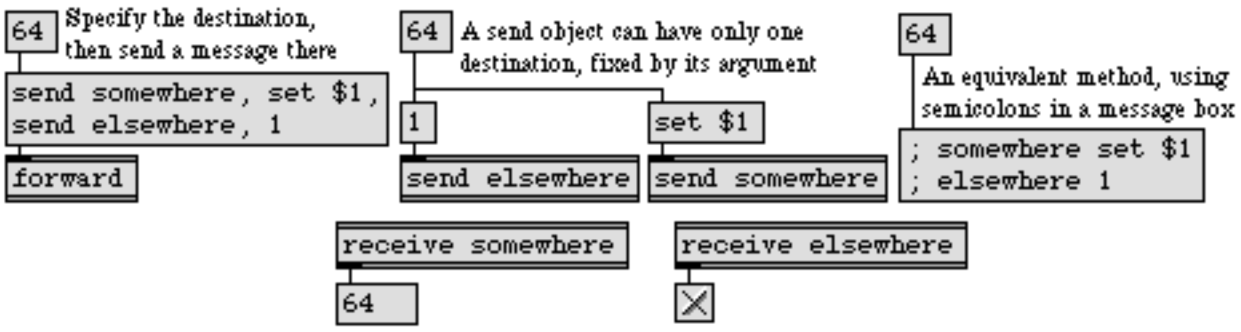
Arguments

any symbol	Optional. Sets the name for the **receive** object which will receive messages. This name can later be changed with the send message.

Output

anything	There are no outlets. A message (other than send) received in the inlet of **forward** is sent out the outlet of each **receive** object of the same name, even if the **receive** is in another patch.

Examples


64
Specify the destination, then send a message there
send somewhere, set $1, send elsewhere, 1
forward
64
A send object can have only one destination, fixed by its argument
1
set $1
send elsewhere
send somewhere
64
An equivalent method, using semicolons in a message box
; somewhere set $1
; elsewhere 1
receive somewhere
64
receive elsewhere

*Using **forward** to send messages to multiple objects at once* — *The same thing, with two **send** objects* — *The **message** box can perform the same function*

See Also

message	Send any message
receive	Receive messages without patch cords
route	Selectively pass the input out a specific outlet
send	Send messages without patch cords
value	Share a stored message with other objects
Tutorial 24	**send** and **receive**

fpic

Display a picture from a graphics file

Note: The **fpic** object requires that QuickTime be installed on your system to open any files other than PICT files. If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.

Input

(mouse) In an unlocked patcher, you can change the offset of the picture by holding down the Shift and Command keys on Macintosh or Shift and Control keys on Windows and dragging on **fpic**; the current offset of the picture is shown in the Assistance portion of the patcher window as you drag.

autoerase The word autoerase, followed by a nonzero number, causes the picture to erase after a new picture is loaded. This mode is disabled by default (autoerase 0).

autofit The word autofit, followed by a nonzero number, scales the graphic to fit in the bounding rectangle of the **fpic** object.

erase The word erase will erase the current picture and then redraw it.

link The word link, followed by symbol which specifies a filename, it will check to see if the graphic has already been loaded by another **fpic** object. If the object has already been loaded into RAM, the **fpic** object will reference the image loaded earlier, conserving memory resources.

matrix The word matrix, followed by nine floating point numbers, reloads the current file into RAM after performing a transformation matrix operation on the image. This transformation is the same one used for the mapping in QuickTime of points from one coordinate space (i.e, the original image) into another coordinate space (a scaled, rotated, or translated version of the original image).

The transform matrix operation consists of nine matrix elements

$$\begin{array}{ccc} a & b & u \\ c & d & v \\ t\_x & t\_y & w \end{array}$$

if *u* and *v* are 0., and *w* is 1., we have the following translation formula.

$x' = a*x + c*y + t\_x;$

$y' = b*x + d*y + t\_y$

The following formulas are used for scaling/rotation:

$a=xscale*cos(\theta)$

$b=yscale*sin(\theta)$

$c=xscale*(-sin(\theta))$

$d=yscale*cos(\theta)$

For more on the transformation matrix, consult the Apple QuickTime Developer documentation found at:
http://developer.apple.com/techpubs/quicktime/qtdevdocs/INMAC/QT/iqMovieToolbox.c.htm#18006

noscale	The word noscale disables image scaling.
offset	The word offset, followed by two numbers, specifies the number of pixels by which the left upper corner of the picture is to be offset horizontally and vertically from the left upper corner of the **fpic** box. By default the left upper corner of the picture is located at the left upper corner of **fpic** (that is, with an offset of 0,0). With successive slightly different offset messages, a picture can be moved inside **fpic**, and **fpic** can window different portions of a large picture. (In order to give the appearance of smooth transitions when moving an image, the old image is not erased when using the offset message. This may cause an undesired appearance if your picture contains a blank background that doesn't cover up what's beneath it.)
pict	The word pict, followed by the name of a graphics file in Max's search path, opens the file and displays the picture, replacing whatever picture was previously displayed. The **fpic** object accepts PICT files and, if QuickTime Version 3.0 or later is installed, other picture file formats that are listed in the QuickTime appendix.
read	The word read, followed by a symbol which specifies a filename, looks for a QuickTime graphic file with that name in Max's file search path, and opens it if it exists, displaying it in a graphic window. If the filename contains any spaces or special characters, the name should be enclosed in double quotes or each special character should be preceded by a backslash (\). The word read by itself puts up a standard Open Document dialog box and displays the common graphics files supported by QuickTime.
readany	The word readany, followed by a symbol which specifies a filename, functions in the same manner as the read message, except that the Open Document dialog box does not filter its display by the currently supported filetypes.
rect	The word rect, followed by four numbers that specify the size of scaling rectangle to apply to fit the input image within, loads the graphics file from disc into RAM and displays it. The first two numbers specify the placement in the graphic window as offset values, and the second two numbers specify the width and height, in pixels, of the rectangle.
scalemode	The word scalemode, followed by number in the range 0-3, sets the scaling mode used by the fpic object.

If the **fpic** object is set to scaling *mode 0*, no scaling is performed; the image is displayed as read into memory.

If the **fpic** object is set to scaling *mode 1*, scaling is performed using the QuickTime transformation matrix (see the matrix message for more information); the image will be scaled and rotated according to the current or default settings of the transformation matrix. The matrix variables can be changed using the **fpic** object's Inspector or by using the matrix message.

If the **fpic** object is set to scaling *mode 2*, rectangular scaling is performed (see the rect message for more information). The image will be loaded and displayed according to the current or default settings of the rect message.

If the **fpic** object is set to scaling *mode 3*, the image is autosized; the **fpic** object scales the graphic to fit in the window currently displayed.

storage — The word storage, followed by two numbers which specify horizontal and vertical distances in pixels, will load only a portion of the graphic image into RAM, which can be used to conserve memory resources.

Note: if either of the arguments are 0, **fpic** will not limit its storage.

time — The word time, followed by a number which specifies a time in QuickTime time units, loads an individual frame from a QuickTime movie and displays it. Typically, QuickTime movies display at a rate of 600 units/second. The default is 0 (i.e., frame one).

Inspector

The behavior of a **fpic** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **fpic** object displays the **fpic** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **fpic** Inspector lets you set the following attributes:

Picture Offset specifies the number of pixels by which the left upper corner of the picture is to be offset horizontally and vertically from the left upper corner of the **fpic** box. By default the left upper corner of the picture is located at the left upper corner of **fpic** (that is, with an offset of 0,0). This offset can be changed by entering new pixel values into the number boxes. The default is no offset (i.e. 0 horizontal, 0 vertical).

Time Offset mode allows you to specify a frame offset in QuickTime time units and load an individual frame of a movie as a graphic. The default is 0 (i.e., frame one).

The *Scaling Mode* pop-up menu can be used to select the type of scaling used by the **fpic** object. There are four scaling modes available: The *None* option (the default) performs no image scaling. Choosing the *Matrix* option will open a patcher window and let you input matrix values for image scaling and rotation. If you have not previously specified matrix values, the defaults will be used. The *Rectangular* option also brings up a patcher window which lets you specify the position of the rectangle within the graphic window, in relative coordinates, and the width and height, in pixels, of the rectangle (the default values are all set to 0). The *Auto-Fit* option will automatically scale the image to fit the display area.

Internal Storage can be used to conserve RAM by only loading a portion of the graphic file into RAM. The area is specified by horizontal and vertical pixel values. Note: if either value is entered as 0, **fpic** will not limit its storage.

The *Picture File* option lets you choose a picture file for the **fpic** object to display by clicking on the Open button. The current file's name appears in the text box to the left of the button. You can also choose a file by typing its name in this box, or by dragging a file icon from the Finder into this box.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

(Get Info…) After placing an **fpic** object in a patcher window, while it is still selected, choose the Get Info… command from the Object menu. This brings up the Inspector window for the fpic object, where you can choose a graphics file to display inside the **fpic** object's box. The picture appears at 100% size, and the **fpic** object's box may then be resized manually to accommodate it. The lower right part of the picture will be cropped by an **fpic** box which is smaller than the size of the picture.

The **fpic** object is simply for displaying pictures in patcher windows. The same visual effect can be achieved by choosing the Paste Picture command from the Edit menu, but that includes the picture in the patcher file, often making the file slow to save and load. Instead, **fpic** just references the graphics file on disk. Another advantage of using the **fpic** object is that it may reduce disk space and memory usage, since the same picture file may be referenced in many patcher windows, rather than being saved in each one. The external graphics file must be in Max's search path, however, in order to be automatically displayed the next time the patch is opened.

Output

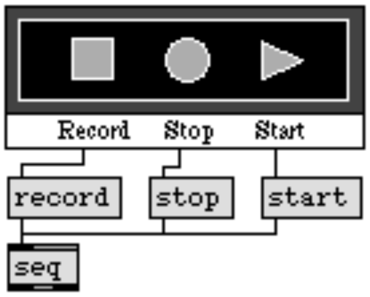
None.

fpic

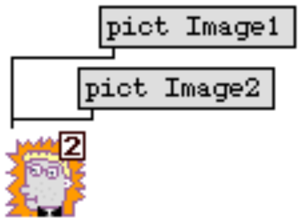
Display a picture from a graphics file

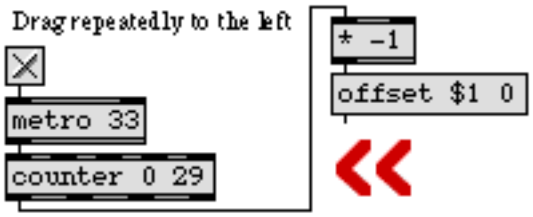
Examples

Place a picture in a patch (for the sheer beauty of it)...



...or make it functional by placing **ubutton** *objects over it.*





Make a slide show by changing pictures, or move a picture by changing its offset

See Also

imovie	Play a QuickTime movie in a patcher window
lcd	Draw graphics in a patcher window
matrixcrtrl	Matrix-style switch control
panel	Colored background area
pictctrl	Picture-based control
pictslider	Picture-based slider
ubutton	Transparent button, sends a bang
Menus	Explanation of commands

frame

Draw framed rectangle in a graphic window

Input

bang — In left inlet: Draws a framed rectangle using the current screen coordinates, drawing mode, and color.

int — In left inlet: Sets the left screen coordinate of the rectangle and draws the shape.

In 2nd inlet: Sets the top screen coordinate of the rectangle.

In 3rd inlet: Sets the right screen coordinate of the rectangle.

In 4th inlet: Sets the bottom screen coordinate of the rectangle.

In 5th inlet: Sets the drawing mode of the rectangle.The following are drawing mode constants; not all modes will be available on all operating systems.

Copy	0	blend	32
Or	1	addPin	33
Xor	2	addOver	34
Bic	3	subPin	35
NotCopy	4	transparent	36
NotOr	5	adMax	37
NotXor	6	subOver	38
NotBic	7	adMin	39

In 6th (right) inlet: Sets the palette index (color) of the frame according to the graphics window's current palette.When the monitor is in black and white mode, any nonzero index is black, and 0 is white.

frgb — In left inlet: The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the color of the frame the next time it is drawn.

priority — In left inlet: The word priority, followed by a number greater than 0, sets a **frame** object's sprite priority in its graphics window. Objects with lower priority will draw behind those with a higher priority.

Arguments

any symbol — Obligatory. The first argument to **frame** must be the name of a graphics window into which the rectangle will be drawn. The window need not exist at the time the **frame** object is created, but the rectangle will not be drawn until the name matches that of an existing and visible window.

int — Optional. Sets the initial sprite priority of the **frame**. If no priority is specified, the default is 3.

frame

Draw framed rectangle in a graphic window

Output

(visual) When the **frame** object's associated graphics window is visible, and a bang message or number is received in its left inlet, a shape is drawn in the window, and the object's previously drawn rectangle (if any) is erased.

Examples

See examples under **oval** or **rect**. **frame** can be directly substituted for **oval**, **rect**, or **ring**.

See Also

graphic	Window for drawing sprite-based graphics
lcd	Draw graphics in a patcher window
oval	Draw solid oval in a graphic window
rect	Draw solid rectangle in a graphic window
ring	Draw framed oval in a graphic window
Graphics	Overview of Max graphics windows and objects

fromsymbol

Transform a symbol into individual numbers or messages

Input

symbol	The **fromsymbol** object accepts a symbol for input, and outputs a list of numbers or messages correspond to the "contents" of the symbol. The **fromsymbol** object is useful for parsing a text symbol composed of numbers, (e.g., "3.5 5 6.5 20") or dividing a symbol up into individual messages.

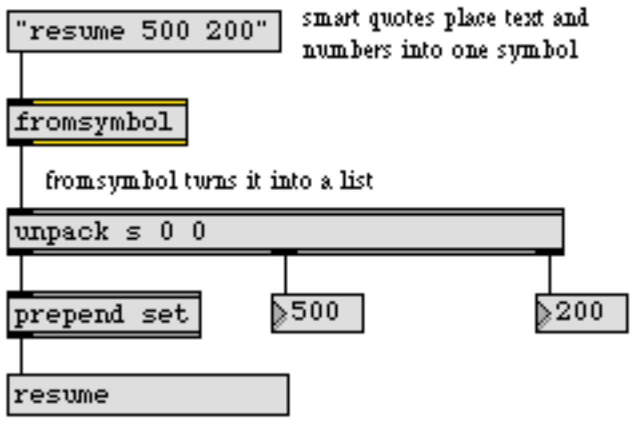
Arguments

None.

Output

messages, lists, or numbers	A list of numbers or messages which correspond to parsed contents of the original symbol.

Examples


"resume 500 200"
smart quotes place text and numbers into one symbol
fromsymbol
fromsymbol turns it into a list
unpack s 0 0
prepend set
500
200
resume

See Also

sprintf	Format a message of words and numbers
tosymbol	Convert messages, numbers, or lists to a single symbol
zl	Multi-purpose list processor

fswap

Reverse the sequential order of two decimal numbers

Input

float	In left inlet: The number is sent out the right outlet, then the number in the right inlet is sent out the left outlet. In right inlet: The number is stored to be sent out the left outlet when a number is received in the left inlet.
int	If there is a float argument, the numbers are converted to float. If there is an int argument or no argument, the number received in the right inlet is stored as an int.
list	In left inlet: The numbers are stored in **fswap**. The first number is sent out the right outlet, then the second number is sent out the left outlet.
bang	In left inlet: Swaps and sends out the numbers currently stored in **fswap**.

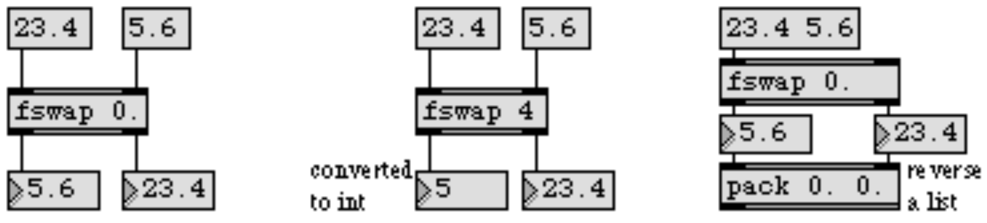
Arguments

int or float	Optional. Sets an initial value for the number which is to be sent out the left outlet. If there is no argument, the initial value is 0. If there is an int argument or no argument, an int is sent out the left outlet. (The number sent out the right outlet is *always* a float.)

Output

int	When a number is received in the left inlet, the number in each inlet is sent out the opposite outlet. If there is an int argument or no argument, an int is sent out the left outlet.
float	The number sent out the right outlet is always a float. The number sent out the left outlet is a float only if there is a float argument.

Examples



Numbers are sent out in reverse order from the order in which they were received

See Also

pack	Combine numbers and symbols into a list
swap	Reverse the sequential order of two numbers
unpack	Break a list up into individual messages

funbuff

Store x,y pairs of numbers together

Input

list	In left inlet: *x* and *y* values for a data pair stored in **funbuff**. If the *x* value is the same as an *x* value already stored in **funbuff**, the previously stored pair is replaced by the new pair.
int	In left inlet: The number is the *x* value of an *x,y* pair. If a *y* value has been received in the right inlet, the two numbers are stored together in **funbuff**. Otherwise, the *x* value causes the corresponding *y* value stored in **funbuff** to be sent out the left outlet. If there is no stored *x* value which matches the number received, **funbuff** uses the closest *x* value which is *less than* the number received, and sends out the corresponding *y* value. In right inlet: The number is a *y* value which will be paired with the next *x* value received in the left inlet, and stored in **funbuff**.
bang	In left inlet: Prints information in the Max window concerning the current status of **funbuff's** contents: how many elements it contains, the minimum and maximum *x* and *y* values it contains, and its domain and range (the maximum minus the minimum, for the *x* and *y* axes respectively).
float	In either inlet: Converted to int.
clear	Erases the contents of **funbuff**.
copy	Copies the current selection (made by using the select message) into the global **funbuff** clipboard. The data stored on this clipboard can then be pasted into another **funbuff** object using the paste message.
cut	Copies the current selection (made by using the select message) into the global **funbuff** clipboard and deletes it from the **funbuff** object. The data stored on this clipboard can then be pasted into another **funbuff** object using the paste message.
delete	In left inlet: The word delete, followed by two numbers, looks for such an *x,y* pair in **funbuff**, and deletes it if it exists. If delete is followed by only one number, only the *x* value is sought, and deleted if it is present.
dump	In left inlet: Sends all the stored pairs out the middle and left outlets in immediate succession. The *y* values are sent out the middle outlet, and the *x* values are sent out the left outlet, in alternation. The pairs are sent out in ascending order based on the *x* value.
embed	The word embed, followed by a non-zero number, causes the **funbuff** data to be stored inside the patcher. The default setting is not to store the **funbuff** data inside the patcher.

find	The word find, followed by a number, will output (out the left outlet) all *x* values (indexes) whose *y* value is equal to the number indicated.
goto	The word goto, followed by a number, sets a pointer to the *x* value (index) specified by the number. A subsequent next message will return the *y* value at the specified x.
interp	In left inlet: The word interp, followed by a number, uses that number as an *x* value, measures its position between its two neighboring *x* values in the **funbuff**, and then sends—out the left outlet—the *y* value that holds a corresponding position between the two neighboring *y* values. If the received number is already the *x* value in a stored *x,y* pair, the corresponding *y* value is sent out. If the received number exceeds the minimum or maximum *x* values stored in **funbuff**, the *y* value that's associated with the minimum or maximum *x* value is sent out. If the **funbuff** is empty, 0 is sent out.
interptab	In left inlet: The word interptab, followed by a number and the name of a named **table** object functions similarly to the interp message (mentioned above), except that it uses the data in the table as an interpolating function. This allows you to easily perform non-linear interpolation between consecutive values in a **funbuff**.
max	Sends the maximum *y* value currently stored in the **funbuff** out the left outlet.
min	Sends the minimum *y* value currently stored in the **funbuff** out the left outlet.
next	Finds the *x* value pointed to by the pointer (or, if the pointer points to a number not yet stored as an *x* value, to the next *greater x* value), and sends the corresponding *y* value out the left outlet. Also, **funbuff** calculates the difference between that *x* value and the value previously pointed to by the pointer, sends the difference out the middle outlet, and resets the goto pointer to the next greater *x* value.
paste	The word paste will copy the contents of the global **funbuff** clipboard into a **funbuff** object. The contents of the clipboard are set using the select, copy and cut messages. These messages provide a handy way of copying data between different **funbuff** objects in any open patchers.
read	Calls up the Open Document dialog box so that a file of *x,y* values can be read into **funbuff**. If the word read is followed by a symbol, Max looks for a file with that name (in the file search path) to load directly into the **funbuff**. The **funbuff** file format is described on the next page.
select	In left inlet: The word select, followed by an two integers representing a starting index and a range will select a region of the **funbuff** which can be edited using the cut, copy and paste messages. For example select 2 3 will select the part of a **funbuff** from index 2 through index 5.

set — In left inlet: The word set, followed by one or more space-separated pairs of numbers, stores each pair as *x,y* pair.

undo — The undo message is used to undo the results of the previous cut or paste message.

write — Calls up the standard Save As dialog box, so that the contents of **funbuff** can be saved as a separate file. If the word write is followed by a symbol, the contents of the **funbuff** are saved immediately in a file, using the symbol as the filename.

Arguments

any symbol — Optional. The argument specifies the name of a file to be read into **funbuff** when the patch is loaded. Changes to the contents of one **funbuff** will not affect the contents of another **funbuff** object with the same name.

A file for **funbuff** can also be created using a text editor window, beginning the text with the word funbuff, followed by a list of space-separated numbers which specify alternating *x* and *y* values. A **funbuff** that has been saved as a file can be viewed and edited as text by choosing Open as Text… from the File menu. Numbers in the form of text can be pasted in from other sources such as the editing window of a **capture** object, or even from another program such as a word processor.

Output

int — Out left outlet: When an *x* value is received in the left inlet, the corresponding *y* value is sent out. (Or, if there is no such *x* value yet stored in **funbuff**, the *y* value corresponding to the next *lesser x* value is sent out.) When the word next is received in the left inlet, **funbuff** sends out the *y* value that corresponds to the *x* value pointed to by its pointer (or, if there is no such *x* value, the *y* value of the next *greater x* value).

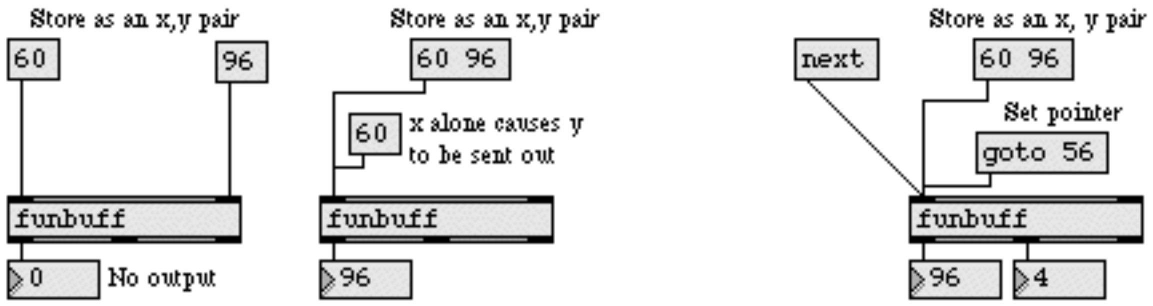
Out middle outlet: When the word next is received in its left inlet, **funbuff** sends out the difference between the *x* value pointed to by its pointer, and the *x* value *previously* pointed to, then resets the pointer to the next *x* value.

bang — Out right outlet: When the pointer reaches the end of a **funbuff**, no numbers are sent out in response to a next message, but a bang is sent out to notify that the end has been reached.

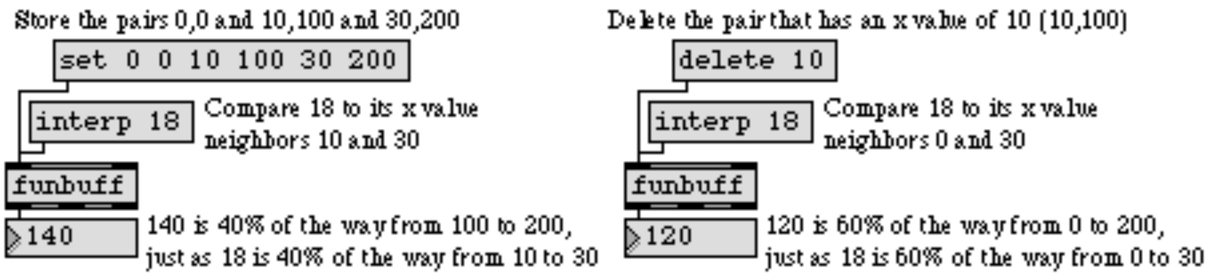
funbuff

Store x,y pairs of numbers together

Examples



Pairs or lists are stored as x,y pairs; an x value alone, or next, *sends out a y value*


Store the pairs 0,0 and 10,100 and 30,200
set 0 0 10 100 30 200
interp 18
Compare 18 to its x value neighbors 10 and 30
funbuff
140
140 is 40% of the way from 100 to 200, just as 18 is 40% of the way from 10 to 30
Delete the pair that has an x value of 10 (10,100)
delete 10
interp 18
Compare 18 to its x value neighbors 0 and 30
funbuff
120
120 is 60% of the way from 0 to 200, just as 18 is 60% of the way from 0 to 30

Interpolating between points stored in **funbuff**

See Also

coll	Store and edit a collection of different messages
envi	Script-configurable envelope in a patcher window
funbuff	Store x,y pairs of numbers together
line	Output numbers in a ramp from one value to another
table	Store and graphically edit an array of numbers
Tutorial 27	Your object
Timeline	Graphically edit a score of Max messages

funnel

Tag data with a number that identifies its inlet

Input

int	In any inlet: The number of the inlet and the received number are sent out as a list.
float	Converted to int.
list	In any inlet: The number of the inlet is prepended to the list, and the new list is sent out. In a list floats are not converted to ints. The list may contain ints, floats, and symbols (provided that the first element of the list is not a symbol).
bang	In any inlet: The number of the inlet and the stored (most recently received) number in that inlet are sent out as a two-item list.

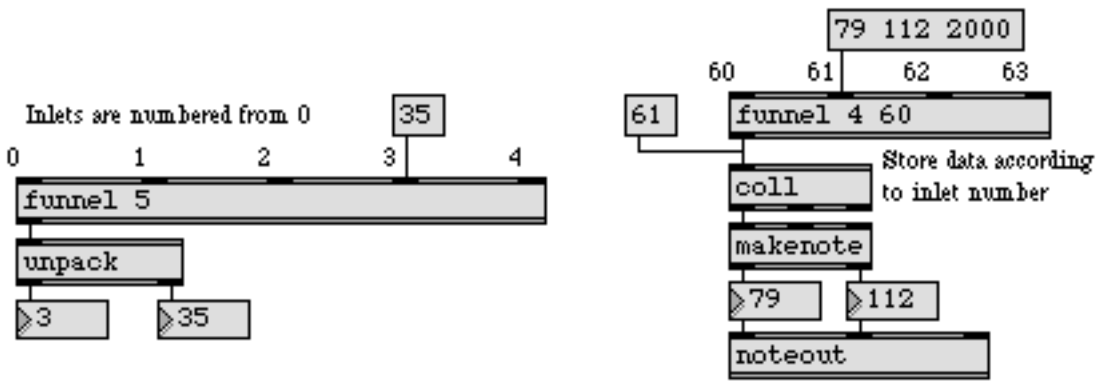
Arguments

int	Optional. The first arguments sets the number of inlets in the **funnel**. If there is no argument there will be two inlets. The second argument specifies an offset for the first inlet number. If no second argument is present, the inlets are numbered beginning with 0.

Output

list	When a number or list is received in any inlet, **funnel** outputs a list consisting of the inlet number followed the input. **funnel** is designed for "funneling" many streams of numbers into the **env** or **envi** objects, but it can be useful in conjunction with other objects such as **coll**, **funbuff** and **table**.

Examples



Use **funnel** *to tag incoming data, or to store data into a* **coll** *object*

See Also

env	Script-configurable envelope editor
envi	Script-configurable envelope in a patcher window
spray	Distribute an integer to a numbered outlet

gate

Pass the input out a specific outlet

Input

int	In left inlet: The number specifies an open outlet through which to pass all messages received in the right inlet. A number in the left inlet does not trigger any output itself.
float	In left inlet: Converted to int.
bang	In left inlet: Reports the current open outlet, or 0 if closed, out the left outlet. This message is designed to be used in conjunction with the **grab** object.
anything	In right inlet: All messages are passed out the open outlet, which is specified by the number in the left inlet.

Arguments

int	Optional. Specifies the number of outlets. Limited between 1 and 10. If there is no argument, there is only one outlet.

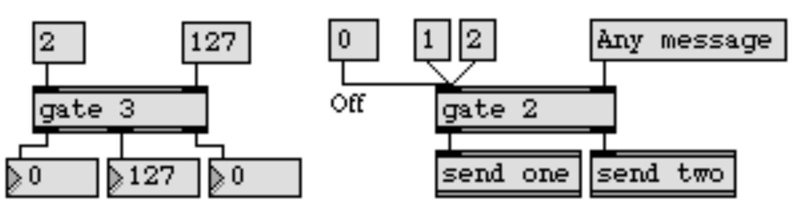
Output

anything	Messages received in the right inlet are passed out the outlet specified by the number in the left inlet. If the number in the left inlet is 0, or if no outlet number has been received yet, all messages are ignored. If the number in the left inlet is less than 0, messages are sent out the *leftmost* outlet. If it is greater than the number of existing outlets, messages are sent out the *rightmost* outlet.

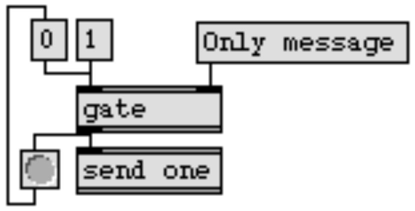
gate

Pass the input out a specific outlet

Examples



Message is passed out the specified outlet



This one closes the door behind itself

See Also

Ggate	Pass the input out one of two outlets
Gswitch	Receive the input in one of two inlets
onebang	Traffic control for bang messages
route	Selectively pass the input out a specific outlet
send	Send messages without patch cords
switch	Output messages from a specific inlet
Tutorial 17	Gates and switches

gestalt

Inquire about current system

Input

various — The **gestalt** object accepts a four-letter symbol specifying a Gestalt selector (a term originating from the Macintosh OS). Examples of useful four-letter codes include sysv for system version and qtim for QuickTime version. For a complete list of Gestalt selectors refer to Apple developer documentation (http://developer.apple.com). On Mac OS, the object uses the Macintosh Gestalt feature to get a response to the selector. On Windows this feature is emulated, and may consequently report slightly different, though meaningful, information.The response and an error code are sent out the object's outlets.

Arguments

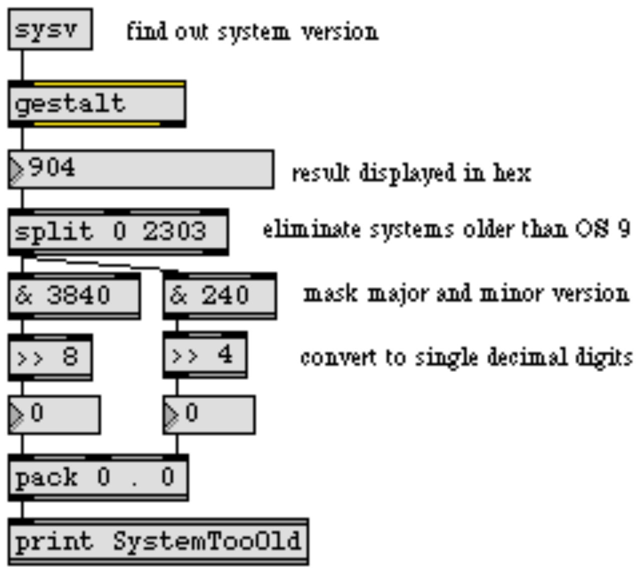
None.

Output

int — Out left outlet If there was no error in obtaining the response to a selector to the object, the response is sent out the left outlet. Binary or hex display and/or the use of the bitwise and operator **&** may aid in interpreting the response.

Out right outlet: If there was an error in obtaining the response to a selector, an error code is sent out the right outlet. Refer to Apple developer documentation for a complete list of error codes. If the input selector was undefined, -1 is sent out. If there was no error, 0 is sent out.

Examples



gestalt *can tell you information about the system in use, plus information about hardware features*

See Also

screensize	Output the monitor size

Ggate

Pass the input out one of two outlets

Input

int	In left inlet: The number specifies which one of the two outlets is to be open. 0 specifies the left outlet, any number other than 0 specifies the right outlet. The arrow on **Ggate** points to the open outlet.
bang	In left inlet: Causes the arrow to point to the other outlet. Clicking on **Ggate** with the mouse has the same effect.
float	In left inlet: Converted to int.
anything	In right inlet: All messages are passed out the open outlet.

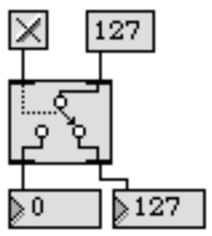
Arguments

None.

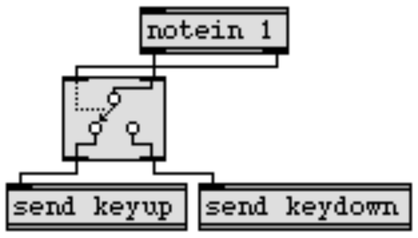
Output

anything	Messages received in the right inlet are passed out one of the two outlets. If the number in the left inlet is 0, incoming messages are sent out the left outlet. If the number in the left inlet is not 0, messages are sent out the right outlet.

Examples



Specify one of two outlets



Any comparison can be used as a criterion

See Also

gate	Pass the input out a specific outlet
Gswitch	Receive the input in one of two inlets
onebang	Traffic control for bang messages
pictctrl	Picture-based control
route	Selectively pass the input out a specific outlet
send	Send messages without patch cords
switch	Output messages from a specific inlet
Tutorial 17	Gates and switches

grab

Intercept the output of another object

Input

anything The message is sent out the right outlet, or if a second argument is present the message is sent to **receive** objects named by the second argument.

set If a second argument has been typed into **grab** specifying the name of a **receive** object, then the word set, followed by a symbol, specifies the name of a (different) **receive** object via which **grab** can grab messages from remote objects.

Arguments

int Optional. The first argument sets the number of outlets, *in addition to* the right outlet. If there is no argument, **grab** has 1 additional outlet.

symbol Optional. If a symbol is present as a second argument, the message received in the inlet is sent to all **receive** objects named by the symbol, instead of being sent out the right outlet. In this case the rightmost outlet, which would normally send out the incoming message if no second argument were present, will not exist.

Output

anything Out right outlet: The right outlet should be connected only to the *leftmost* inlet of other objects. The message received in the inlet is sent out to the left inlet of all objects connected to the right outlet. Whatever goes out their outlets, however, is then *intercepted* by **grab**.

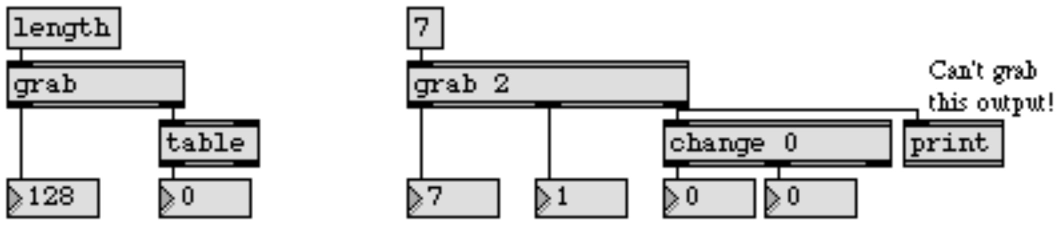
Out other outlets: Whatever would normally be sent out the outlets of the objects connected to the right outlet, is sent out **grab's** outlets instead, in response to a message from **grab**. Whatever would be sent out the leftmost outlet of the other objects is sent out the leftmost outlet of **grab**, and so on. Note: Only the output that is sent out the *outlets* of other objects can be intercepted by **grab**. Other types of output, such as transmission of MIDI messages or printing in the Max window, cannot be intercepted by **grab**. Also, **grab** does not intercept the output of timing objects such as **seq**, **metro**, and **clocker**.

Connecting the right outlet of **grab** to the inlet of a patcher object, however, will *not* grab the output of the subpatch. It will simply grab the output of the **inlet** object inside the subpatch, which is exactly the same as its input. However, **grab** can communicate with remote objects via a **receive** object named as the second argument to **grab**.

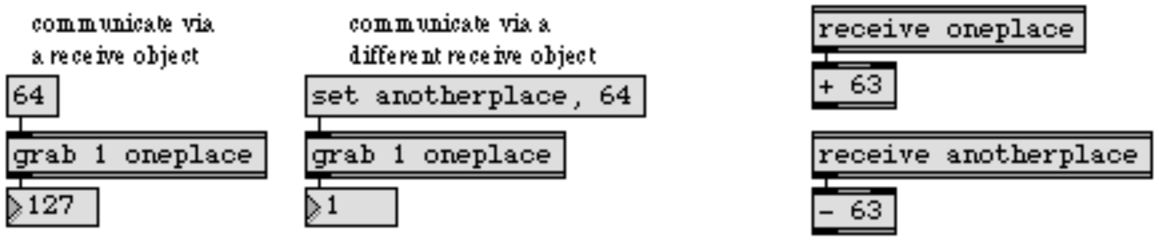
If a second argument is present, the message received in the inlet is sent directly to **receive** objects named by the argument instead of being sent out the right outlet. Any such **receive** objects should be connected only to *the leftmost inlet* of other objects. The rightmost outlet, which would otherwise be used to grab the output of other objects, does not appear if the second argument is used.

Note that if **grab** is connected to other objects remotely via numerous **receive** objects of the same name, the order in which **grab** communicates with those other objects is undefined, so the order in which their output will be sent out of the **grab** object's other outlets is unpredictable.

Examples



Get an object's output by "grabbing" it before it comes out the outlet



grab *can communicate with any* **receive** *object specified by a* set *message*

See Also

preset	Store and recall the settings of other objects
table	Store and graphically edit an array of number

graphic

Window for drawing sprite-based graphics

Input

open — Causes the graphics window associated with the **graphic** object to become visible. The window is also brought to the front. Double-clicking on the **graphic** object in a locked patcher has the same effect.

wclose — Causes the window associated with the **graphic** object to become invisible.

Arguments

symbol — Optional. Identifies the **graphic** object's window. Drawing and animation objects use this symbol to tell Max which window to draw in. If no argument is typed in, the window will be named *Graphics—1* (and subsequent graphics windows will be numbered sequentially).

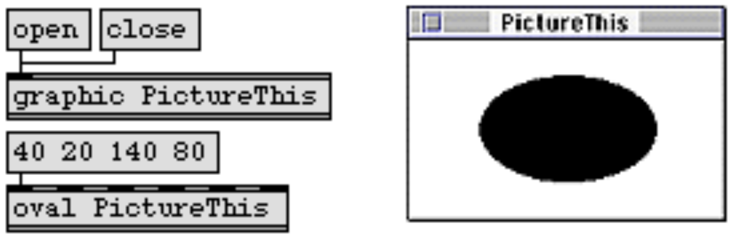
int — Optional. Following the name of the **graphic** object, four coordinates can be specified for the location of the window on the screen. The numbers represent the screen coordinates of the left, top, right, and bottom corners (respectively) of the drawing area. Note that when you save a patch containing a **graphic** object with no coordinate arguments, the current window location is saved. The coordinate arguments are useful in the case where you want the object's window to be guaranteed to appear in a certain position each time the patch is opened, regardless of where it may have been dragged in the past.

Optional. Following the name of the **graphic** object, but preceding the four coordinate arguments, a fifth non-zero number argument may be inserted, which will cause the graphics window's title bar to be hidden. A graphics window without a title bar can still be dragged by Command-clicking on it on Macintosh or Control-clicking on Windows.

Output

None. Other objects draw into a **graphic** object's window.

Examples



The **graphic** *object creates a window for the output of graphics objects. The window can be resized by dragging in the lower right corner where you'd expect the grow box to be.*

See Also

frame	Draw framed rectangle in a graphic window
graphic	Window for drawing sprite-based graphics
lcd	Draw graphics in a patcher window
oval	Draw solid oval in a graphic window
pict	Draw picture in a graphic window
rect	Draw solid rectangle in a graphic window
ring	Draw framed oval in a graphic window
Graphics	Overview of Max graphics windows and objects
Tutorial 42	Graphics

Gswitch

Receive the input in one of two inlets

Input

int	In left inlet: The number specifies which one of the other two inlets is to be open. 0 specifies the middle inlet, any number other than 0 specifies the right inlet. The arrow on **Gswitch** points to the open inlet.
bang	Causes the arrow to point to the other inlet. Clicking on **Gswitch** with the mouse has the same effect.
float	In left inlet: Converted to int.
anything	In middle or right inlet: Messages received in the open inlet are passed out the outlet, while messages received in the other inlet are ignored.

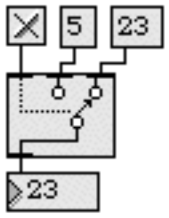
Arguments

None.

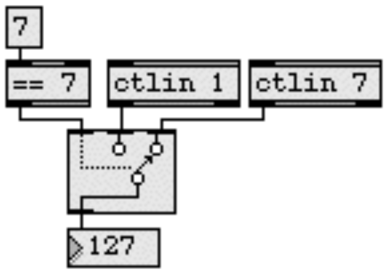
Output

anything	If the number in the left inlet is 0, all messages received in the middle inlet are passed out the outlet, and messages received in the right inlet are ignored. If the number in the left inlet is *not* 0, messages received in the middle inlet are ignored, and all messages received in the right inlet are passed out the outlet.

Examples



Specify one of two inlets



Any comparison can be used as a criterion

See Also

gate	Pass the input out a specific outlet
Ggate	Pass the input out one of two outlets
pictctrl	Picture-based control
receive	Receive messages without patch cords
route	Selectively pass the input out a specific outlet
switch	Output messages from a specific inlet
Tutorial 17	Gates and switches

hint

Pop-up style hint text

Input

(mouse)	When the cursor moves within the **hint** object's rectangle, its text message will appear in a colored area beneath the rectangle after the specified delay.
(Font menu)	The appearance of the **hint** object can be altered by selecting it and choosing a different font or size from the Font menu.
delay	The word delay, followed by a number, sets the delay in milliseconds until the hint appears. The default is 1000 (i.e., one second).
brgb	(Windows only) The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **hint** object. The default value is white (brgb 255 255 255).
frgb	(Windows only) The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the text displayed by the **hint** object. The default value is black (frgb 0 0 0).
set	The word set, followed by any message, will replace the message stored in **hint**. This message will be displayed when the mouse is positioned over the **hint** object after an interval of time specified by the delay message.

Inspector

The behavior of a **hint** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **hint** object displays the **hint** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **hint** Inspector lets you set the following attributes:

Type the text you want displayed when the mouse is positioned over the area bounded by the hint object into the *Set Hint Text* box.

The *Pop-up Delay* lets you set the delay in milliseconds until the hint appears. The default is 1000 (one second).

Check Interval sets the interval in milliseconds at which the mouse position is checked. The default is 100.

If the *Redraw Behind Hint* checkbox is checked, anything in the patcher window which is underneath the hint will be erased and redrawn. This mode should be used if the hint message will appear, in an area over something which could change its appearance while the hint is visible (i.e., a number box or a slider). The default is on (checked).

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

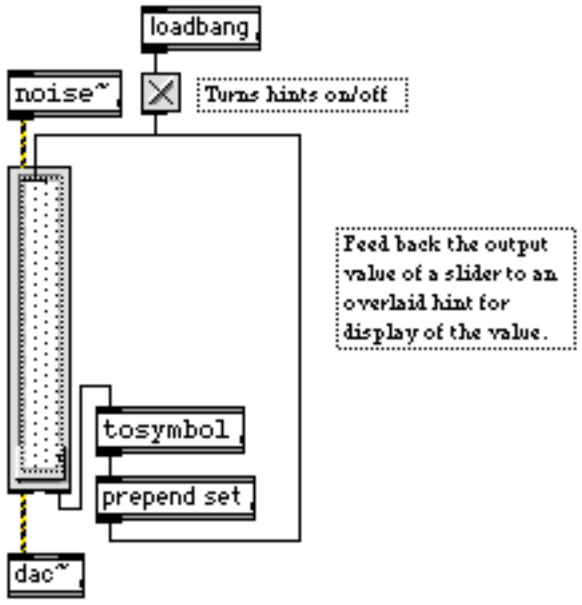
Arguments

None.

Output

message	The message stored in the **hint** object.

Examples


loadbang
noise~
Turns hints on/off
Feed back the output value of a slider to an overlaid hint for display of the value.
tosymbol
prepend set
dac~

Provide optional hints to UI objects

See Also

comment	Explanatory note or label
umenu	Pop-up menu, to display and send commands

histo

Make a histogram of the numbers received

Input

int — In left inlet: **histo** keeps count of how many times it has received a number between 0 and 127 in the left inlet. When a number is received, **histo** includes it in the count, sends the number of times that number has been received out the right outlet, and passes the number itself out the left outlet. Numbers outside the range 0-127 are ignored.

In right inlet: Has the same effect as a number in the left inlet, except that the number is not counted by **histo**.

clear — Erases the memory of **histo**, to begin a new histogram.

bang — In left inlet: Using the number most recently received in the left inlet, **histo** reports out the right outlet how many times that number has been received, and sends the number itself out the left outlet. If no number has been previously received in the left inlet, 0 is sent out both outlets.

Arguments

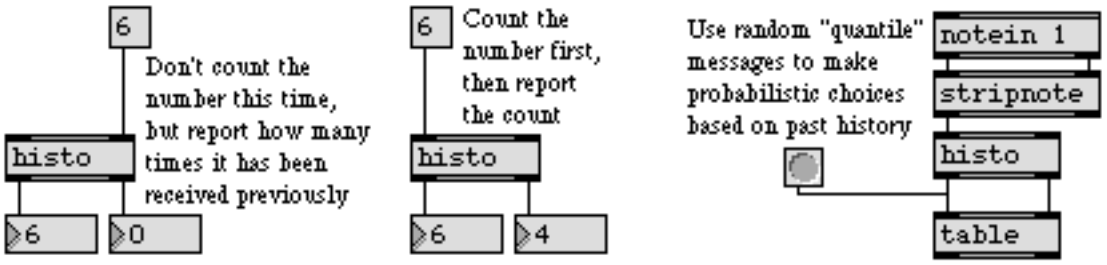
None.

Output

int — Out left outlet: The number received in the inlet.

Out right outlet: The count of the number of times that number has been received.

Examples


6
Don't count the number this time, but report how many times it has been received previously
histo
6
0
6
Count the number first, then report the count
histo
6
4
Use random "quantile" messages to make probabilistic choices based on past history
notein 1
stripnote
histo
table

Store a histogram of the numbers received; display it in a **table**

See Also

anal	Make a histogram of number pairs received
prob	Make weighted random series of numbers
table	Store and graphically edit an array of numbers
Tutorial 33	Probability tables
Quantile	Using **table** for probability distribution

hslider

Output numbers by moving a slider onscreen

Input

int	The number received in the inlet is displayed graphically by **hslider**, and is passed out the outlet. Optionally, **hslider** can multiply the number by some amount and add an offset to it, before sending the number out its outlet. The **hslider** will also send out numbers in response to mouse clicking or dragging.
float	Converted to int.
bang	Sends out the number currently stored in **hslider**.
color	The word color, followed by a number from 0 to 15, sets the color of the center portion of the **hslider** to one of the object colors which are also available via the Color command in the Object menu.
local	The word local, followed by a non-zero number, enables object response to mouse clicks (the default). The message local 0 disables the object's response to the mouse; the **hslider** object will respond only to input in its inlet and ignore all mouse clicks.
min	The word min, followed by a number, sets value that will be added to the **hslider** object's value before it is sent out the outlet. The default is 0.
mult	The word mult followed by a number, specifies a multiplier value. The **hslider** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default value is 1.
resolution	The word resolution, followed by a number, sets the sampling interval in milliseconds. This controls the rate at which the display is updated as well as the rate that numbers are sent out the **hslider** object's outlet.
set	The word set, followed by a number, resets the value displayed by **hslider**, without triggering output.
size	The word size, followed by a number, sets the range of the **hslider** object. The default value is 128. Setting the size to 1 disables the **hslider** visually (since it can only display one value). Any specified size less than 1 will be set to 2.

Inspector

The behavior of an **hslider** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **hslider** object displays the **hslider** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

hslider

Output numbers by moving a slider onscreen

The **hslider** Inspector lets you enter a *Slider Range* value. Numbers received in the inlet are automatically limited between 0 and the number 1 less than the specified range value. The default range value is 128. You can specify an *Offset* value which will be added to the number, after multiplication. The default offset value is 0. The **hslider** Inspector also lets you specify a *Multiplier*. The **hslider** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default multiplier value is 1.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

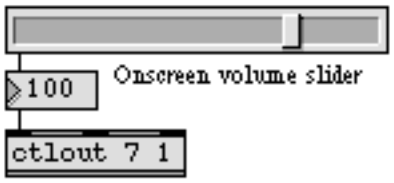
Arguments

None.

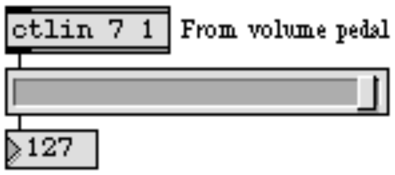
Output

int	Numbers received in the inlet, or produced by clicking or dragging on **hslider** with the mouse, are first multiplied by the *multiplier*, then have the *offset* added to them, then are sent out the outlet.

Examples



Produce output by dragging onscreen...



or use to display numbers passing through

See Also

kslider	Output numbers from a keyboard onscreen
multislider	Multiple slider and scrolling display
pictctrl	Picture-based control
pictslider	Picture-based slider
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

Input

int	The number in each inlet will be stored in place of the $i or $f argument associated with it. (Example: The number in the second inlet from the left will be stored in place of the $i2 and $f2 arguments, wherever they appear.)
float	The number in each inlet will be stored in place of the $f or $i argument associated with it. The number will be truncated by a $i argument.
symbol	In left inlet: The word symbol, followed by a symbol (a word), will be stored in place of the $s1 argument.
bang	In left inlet: Evaluates the conditional statement using the values currently stored.
	Any of the above messages in the left inlet will evaluate the conditional statement and send out the result. Any inlets which have not yet received a value have the value 0 by default.
	The number of inlets is determined by how many different changeable arguments are typed in. The maximum number of inlets is 9.
list	In left inlet: The items of the list are treated as if each had come in a different inlet, and the conditional statement is evaluated. If the list contains fewer items than there are inlets, the most recently received value in each remaining inlet is used.
set	In left inlet: The word set, followed by one or more numbers, treats those numbers as if each had come in a different inlet, replacing the stored value with the new value, but the conditional statement is not evaluated and nothing is sent out the outlet. If there are fewer numbers in the message than there are inlets, the stored value in each remaining inlet is left unchanged.

Arguments

Obligatory. The arguments for the **if** object start with a *conditional statement* that uses the same syntax as **expr**. Refer to the description of the **expr** object for details. The word then follows the conditional statement, which is then followed by a message expression described below. After the message expression, there is an optional else and a second message expression.

if evaluates the conditional expression, and if the result is non-zero, evaluates the message expression after the word then. Otherwise, it evaluates the second message expression after the word else (or does nothing in the case where no else and second message expression have been typed in.

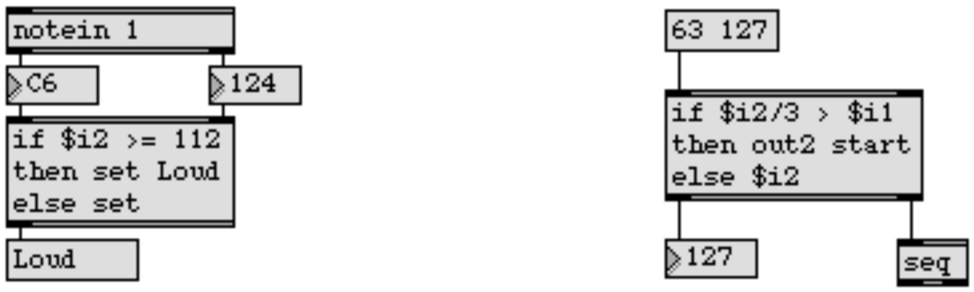
then, else	Message expressions are similar to what you type into a **message** box, with the following differences:
$i1,$f1,$s1	You use $i1, $f1, or $s1 instead of $1 for changeable arguments.

send	No commas or semicolons are allowed. Messages can be sent to remote **receive** objects by preceding the message expression with **send**, followed by the name of the **receive** object.
out2	The keyword out2 in a message expression creates a second, right outlet for the **if** object. If out2 precedes a message expression, the result of the expression is sent out the right outlet instead of the left outlet.

Output

anything	The message after the then or else portion of the arguments is sent out the outlet. If the word out2 is present as an argument, there will be two outlets, and messages following out2 will be sent out the right outlet. If the word send is present as an argument, the word that follows it is the name of a **receive** object, and the message that follows it will be sent to **receive** objects with that name.

Examples


notein 1
C6
124
if $i2 >= 112 then set Loud else set
Loud
63 127
if $i2/3 > $i1 then out2 start else $i2
127
seq

Complex comparisons and results can be described in a single object

See Also

!=	Compare two numbers, output 1 if they are not equal
<	*Is less than*, comparison of two numbers
<=	*Is less than or equal to*, comparison of two numbers
==	Compare two numbers, output 1 if they are equal
>	*Is greater than*, comparison of two numbers
>=	*Is greater than or equal to*, comparison of two numbers
expr	Evaluate a mathematical expression
select	Select certain inputs, pass the rest on
Tutorial 38	**expr** and **if**

imovie

Play a QuickTime movie in a patcher window

Note: The **imovie** object requires that QuickTime be installed on your system. If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.

Input

(see **movie**)	All messages recognized by the **movie** object are similarly recognized by **imovie**.
border	The object is initially shown with a black line border drawn around its movie. The message border 0 erases the black line border; border 1 redraws the border.

Arguments

(Get Info…)	Optional. Selecting the object (when the patcher window is unlocked) and choosing the Get Info… command from the Object menu opens a standard file dialog, allowing you to select a QuickTime movie to be read into the object automatically when the patch is loaded. The movie must be located in Max's file search path (specified with the File Preferences… command in the Options menu) in order for **imovie** to find it automatically.

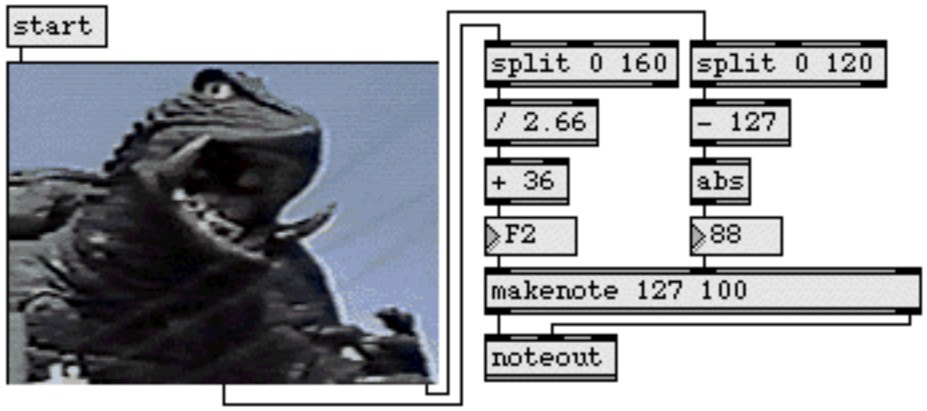
Output

int	Out left outlet: The end time of the movie is sent out in response to the length message; the current time in the movie is sent out in response to the time message; 0 is sent out in response to the start message.
	Out middle outlet: The horizontal position of the mouse, relative to the left edge of the movie, is sent out when the mouse is clicked or dragged inside the movie.
	Out right outlet: The vertical position of the mouse, relative to the top edge of the movie, is sent out when the mouse is clicked or dragged inside the movie.

imovie

Play a QuickTime movie in a patcher window

Examples



A movie can be displayed within a patch, and mouse motion can be detected within it

See Also

lcd	Draw graphics in a patcher window
movie	Play a QuickTime movie in a window
playbar	QuickTime movie play controller

IncDec

Buttons that increment/decrement a value

Input

int	A number sent to the **IncDec** object's inlet sets the value that will be incremented or decremented by clicking on the top or bottom of half of the object. The number is not sent out the outlet. **IncDec** is designed to be used with user interface objects such as the **number box**, **dial**, and the various sliders.
(mouse)	A mouse click increments or decrements the stored value (depending on which arrow is clicked) and sends it out the outlet.
(Font menu)	The height of an **IncDec** object can be altered by selecting it and choosing a different font or size from the Font menu.

Arguments

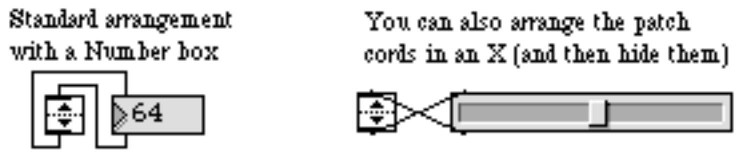
None.

Output

int — When you click on the top half of an **IncDec** object, it sends out a value that is one greater than the last value received at its inlet or sent out its outlet, whichever happened most recently. Holding the mouse button down continues to increment the output, gradually increasing in rate of output.

The same is true for the bottom half of **IncDec**, except that the values are *decremented*.

Examples


Standard arrangement
with a Number box
64
You can also arrange the patch
cords in an X (and then hide them)

IncDec *works well in combination with* **number box** *and* **hslider**

See Also

counter	Count the bang messages received, output the count
number box	Display and output a number
hslider	Output numbers by moving a slider onscreen
umenu	Pop-up menu, to display and send commands
uslider	Output numbers by moving a slider onscreen

inlet

Receive messages from outside a patcher

Input

(patcher) Each **inlet** object in a patcher will show up as an inlet at the top of an object box when the patch is used inside another patcher (as an object or a subpatch). Messages sent into such an inlet will be received by the **inlet** object in the subpatch. A patcher can have a maximum of 250 signal inlets. The number of data inlets is a much bigger number than that.

Inspector

A descriptive Assistance message can be assigned to an **inlet** object and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **inlet** object displays the **inlet** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

Typing in the *Describe Outlet* text area specifies the content of the Assistance message.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

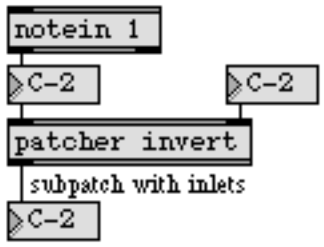
Output

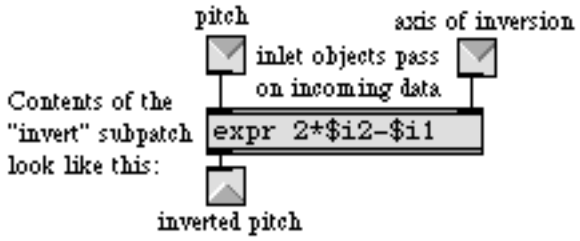
anything In a subpatch **inlet** sends out whatever messages it receives through patch cords from the patch that contains it.

inlet

Receive messages from outside a patcher

Examples





pitch
axis of inversion
inlet objects pass on incoming data
Contents of the "invert" subpatch look like this:
expr 2*$i2-$i1
inverted pitch

Inlets of the subpatch... *correspond to the* **inlet** *objects in the subpatch*

See Also

bpatcher	Embed a visible subpatch inside a box
outlet	Send messages out of a patcher
pcontrol	Open and close subwindows within a patcher
receive	Receive messages without patch cords
send	Send messages without patch cords
Tutorial 26	The **patcher** object

int / i

Store an integer value

Input

int	In left inlet: The number replaces the currently stored value and is sent out the outlet. In right inlet: The number replaces the stored value without triggering output.
float	Converted to int.
bang	In left inlet: Sends the stored value out the outlet.
set	In left inlet: The word set, followed by a number, replaces the stored value without triggering output.
send	In left inlet: The word send, followed by the name of a **receive** object, sends the value stored in **int** to all **receive** objects with that name, without sending it out the outlet of the **int**.

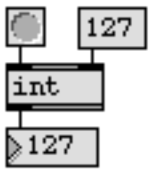
Arguments

int	Optional. Sets an initial value to be stored in **int**. If there is no argument, the initial value is 0. An int argument by itself, without the word int, is another way of creating and initializing an **int** object.
float	Converted to int.

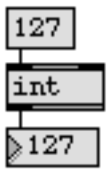
Output

int	A number is stored in (and output from) **int** as a long (32-bit) integer.

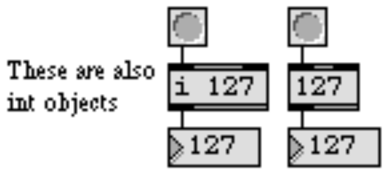
Examples



Output the stored value



Replace the stored value and output it



Initial value is given

See Also

float	Store a decimal number
pv	Share variables specific to a patch and its subpatches
value	Share a stored message with other objects
Tutorial 21	Storing numbers

iter

Break a list up into a series of numbers

Input

list	The numbers in the list are sent out the outlet in sequential order.
int or float	The number is sent out the outlet.
bang	Sends the number or list most recently received, in sequential order.

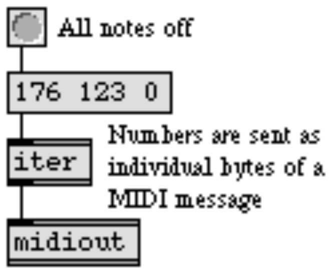
Arguments

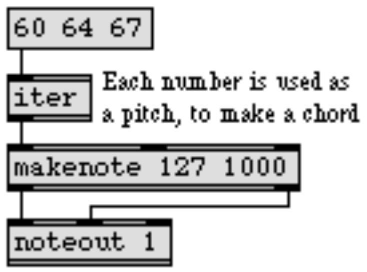
None.

Output

int	The numbers received in the inlet are sent out one at a time.

Examples





Numbers in a list pass through **iter** *one at a time*

See Also

cycle	Send a stream of data to individual outlets
thresh	Combine numbers into a list, when received close together
unpack	Break a list up into individual messages
zl	Multi-purpose list processor
Tutorial 30	Number groups

key

Report key presses on the computer keyboard

Input

(keyboard) The input to **key** comes directly from the computer keyboard. There are no inlets.

Arguments

None.

Output

int Output is sent each time a key is depressed on the computer keyboard. (Holding the key down does not produce repeated output.)

Out left outlet: The ASCII value of the typed key.

Out middle outlet: The key code of the typed key.

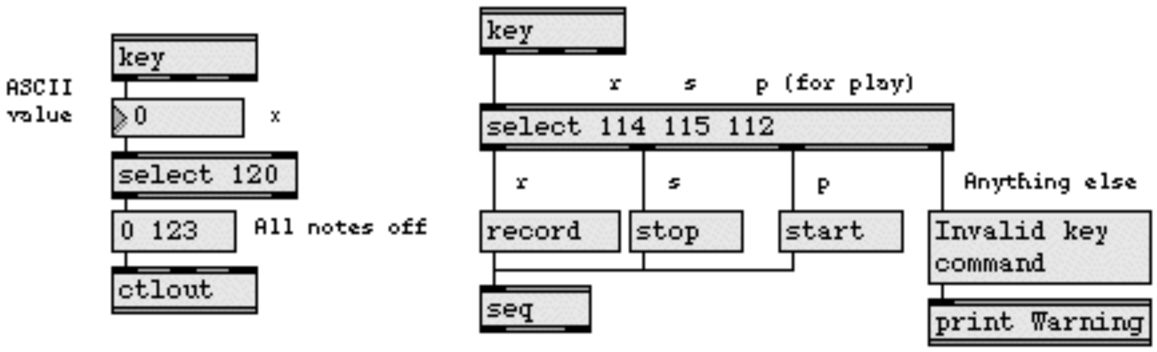
Out right outlet: The output values can be sent through the **&** object to create toggles set by each modifier key. The numerical output of the right outlet is listed below along with the argument to the **&** object that will create a toggle.:

Modifier Key	*Output*	*Toggle*
key events	128	& 128 (reports 0 on Windows if a mouse button is down, always reports 0 on Macintosh
Windows Control key	384	& 256 (system uses this so it is not reported)
Macintosh Command key	384	& 256 (system uses this so it is not reported)
Shift key	640	& 512
Caps Lock key (on)	1152	& 1024
Windows Alt key	2176	& 2048 (on Windows the system uses this so it is not reported)
Macintosh Option key	2176	& 2048
Windows R. Mouse Button	4224	& 4096
Macintosh Control key	4224	& 4096

key

Report key presses on the computer keyboard

Examples



Keys typed on the computer keyboard can be used to trigger messages

See Also

keyup	Report key releases on the computer keyboard
numkey	Interpret numbers typed on the computer keyboard
spell	Convert input to ASCII codes
sprintf	Format a message of words and numbers
Tutorial 20	Using the computer keyboard

keyup

Report key releases on the computer keyboard

Input

(keyboard) The input to **keyup** comes directly from the computer keyboard. There are no inlets.

Arguments

None.

Output

int Output is sent each time a key is released on the computer keyboard. (Nothing is sent when the key is first depressed.)

Out left outlet: The ASCII value of the typed key.

Out right outlet: The key code of the typed key.

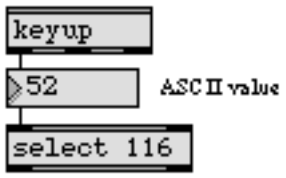
Out right outlet: The output values can be sent through the **&** object to create toggles set by each modifier key. The numerical output of the right outlet is listed below along with the argument to the **&** object that will create a toggle.:

Modifier Key	*Output*	*Toggle*
key events	128	& 128 (reports 0 on Windows if a mouse button is down, always reports 0 on Macintosh
Windows Control key	384	& 256 (system uses this so it is not reported)
Macintosh Command key	384	& 256 (system uses this so it is not reported)
Shift key	640	& 512
Caps Lock key (on)	1152	& 1024
Windows Alt key	2176	& 2048 (on Windows the system uses this so it is not reported)
Macintosh Option key	2176	& 2048
Windows R. Mouse Button	4224	& 4096
Macintosh Control key	4224	& 4096

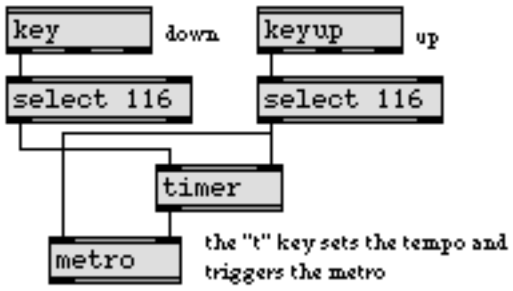
keyup

Report key releases on the computer keyboard

Examples



ASCII value is sent when key is released



Used with **key** *to measure how long a key is down*

See Also

key	Report key presses on the computer keyboard
mousestate	Report the status and location of the mouse
numkey	Interpret numbers typed on the computer keyboard
spell	Convert input to ASCII codes
sprintf	Format a message of words and numbers
Tutorial 20	Using the computer keyboard

kslider

Output numbers from a keyboard onscreen

Input

int	In left inlet: The number received in the inlet is displayed graphically by **kslider** if it falls within its displayed range. The current velocity value (from 1 to 127) that **kslider** holds is sent out its right outlet, followed by the received number out the left outlet. In right inlet: The number received in the right inlet sets the output key velocity without triggering output.
(mouse)	**kslider** also sends out numbers when you click or drag on it with the mouse. The velocity value is determined by the vertical position of the mouse within each key. Higher vertical positions produce higher velocities, to a maximum of 127. If the **kslider** object is in polyphonic mode, you need to click on a key twice: once to send a note-on, and once again for a note-off. Clicking on the very rightmost edge of the **kslider** sends out the note of the key C that would be just to the right of the keys that are visible.
float	Converted to int.
bang	In left inlet: Sends out the pitch and velocity values currently stored in **kslider**.
chord	In left inlet: The word chord, followed by a list of MIDI note name and velocity pairs, can be used to play chords on the **kslider** in polyphonic mode (set by the mode 1 message). The chord message sends note-offs for currently held notes, followed by note-on commands for the specified note and velocity pairs. When the **kslider** object's state is saved by a **preset** object in polyphonic mode, the **preset** object will store chord messages.
clear	In left inlet: The clear message will clear any currently highlighted notes on the keyboard, but will not trigger any output.
color	In left inlet: The word color, followed by a number from 0 to 15, sets the color of the keyboard that is highlighted to one of the object colors that are also available with the Color submenu of the Object menu.
flush	In left inlet: When the **kslider** object is in polyphonic mode (set by the mode 1 message), the flush message will send note-offs to currently held notes and clear the **kslider** object's display.
frgb	In left inlet: The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the color of the part of the keyboard that is highlighted (default 128 128 128).
mode	In left inlet: The word mode, followed by a 0 or 1, selects monophonic or polyphonic operation for the kslider. mode 0 (default) sets monophonic mode. Only

one key can be selected and displayed at one time. mode 1 sets the **kslider** to polyphonic mode. In polyphonic mode, **kslider** keeps track of note-ons and note-offs, so it mirrors which notes are currently held down on your MIDI keyboard. A key is "turned off" by sending the **kslider** object a key on message with a velocity of 0.

offset In left inlet: The word offset, followed by a number, sets an offset value in octaves for the kslider object. The default kslider keyboard outputs notes from the lowest octave of the MIDI keyboard range (c-2). The message offset 5 would mean that the kslider object's leftmost key would be C3. The default is 3.

range In left inlet: The word range, followed by a number, sets the range of the **kslider** object in octaves. The default value is 4.

set In left inlet: The word set, followed by a number, changes the value displayed by **kslider**, without triggering output.

size In left inlet: The word size, followed by a 0 or 1, sets the size of the keyboard display. size 0 (default) sets the large keyboard, and key 0 selects the small keyboard.

Inspector

The behavior of a **kslider** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **kslider** object displays the **kslider** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **kslider** Inspector lets you enter a *Slider Range* value (default 4) that sets the range of the **kslider** object in octaves. An *Offset* value (default 3) specifies the number of octaves the lowest note on the displayed keyboard will from C-2 (the lowest MIDI C). the *Keyboard Size* buttons select the size of the keyboard, and the *Keyboard Mode* buttons select monophonic or polyphonic modes. The *Color* option lets you use a swatch color picker or RGB values to specify the color of the highlighted portion of the keyboard. The default color is 128 128 128.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

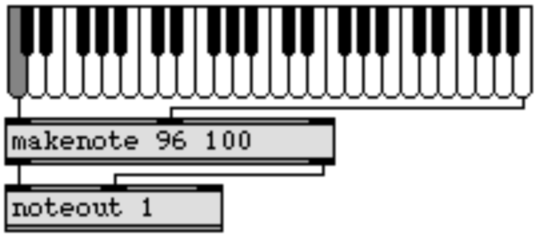
kslider

Output numbers from a keyboard onscreen

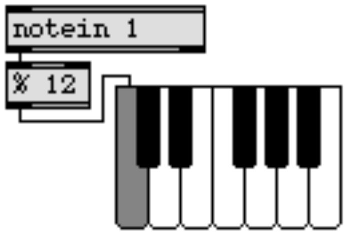
Output

int **kslider** sends its current velocity value out its right inlet, followed by the (displayable) pitch value out its left outlet, when a number is received in its inlet or you click or drag on the object.

Examples



Produce output by clicking on the keyboard...



or use to display incoming pitches

See Also

hslider	Output numbers by moving a slider onscreen
makenote	Generate a note-off message following each note-on
notein	Output received MIDI note messages
noteout	Transmit MIDI note messages
pictslider	Picture-based slider
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

lcd

Draw graphics in a Patcher window

In Max 4.0 and later, all **lcd** object drawing commands are now lower case. For backwards compatibility, old style capitalized message names are still understood; you can use either lineto or LineTo.

Input

(mouse)	You can draw freehand in **lcd** with the mouse (provided this feature has not been turned off with a local 0 message). The mouse will draw with the current pen and color characteristics, and the mouse location will be sent out the outlet.
ascii	The word ascii, followed by a number between 0 and 255, writes the character corresponding to that ASCII value at the current pen position, then moves the pen position to the right of that character. Numbers that exceed the 0-255 range are restricted to that range with a modulus operation.
backsprite	The word backsprite, followed by a symbol, sets the named sprite's drawing order so that it is drawn first (and displayed last). This command can be used to alter the order in which sprites are drawn. (Normally, sprites are drawn in the order they are recorded.)
border	border 1 sets **lcd** to draw a border around its window, which is on by default. A message of border 0 turns this feature off.
brgb	The word brgb, followed by three numbers between 0 and 255, specify an RGB value sets the current background color of the **lcd** object.
clear	Erases the contents of **lcd**.
clearpicts	Deletes all of an **lcd** object's named pictures.
clearregions	Deletes all of an **lcd** object's named regions.
clearsprites	Deletes all of an **lcd** object's named sprites.
clipoval	followed by four int arguments specifying the left, top, right, and bottom extremities of an oval, clips drawing commands to the oval. These extremities are specified in pixels, relative to the top left corner of the **lcd** display area.
clippoly	The word clippoly may be followed by as many as 254 int arguments that would specify a series of x/y pairs that define a polygon to which **lcd** will clip drawing commands. These x/y pairs are specified in pixels, relative to the top left corner of the **lcd** display area.
cliprect	The word cliprect, followed by four int arguments specifying the left, top, right, and bottom positions of a rectangle, clips **lcd** drawing commands to the rectangle. These edge positions are specified in pixels, relative to the top left corner of the **lcd** display area.

cliprgn	The word cliprgn, followed by a symbol, clips drawing commands with the named region.
cliproundrect	he word cliproundrect, followed by six int arguments specifying the left, top, right, and bottom positions of a rectangle and the amount of horizontal and vertical roundness in pixels, clips drawing commands to a rounded rectangle. The edge positions are specified in pixels, relative to the top left corner of the **lcd** display area.
closeregion	The word closeregion, followed by a symbol argument that names the region, turns off region definition and associates the defined region with the symbol. After the closeregion message, drawing commands function normally again.
closesprite	The word closesprite, followed by a symbol argument that names the sprite, turns off sprite command collection and associates the defined region with the symbol. After the closesprite message, drawing commands function normally again.
color	The word color, followed by a number from 0 to 255, specifies a color (from Max's color palette) for subsequent graphics drawn in **lcd**. Numbers that exceed the 0-255 range are restricted to that range with a modulus operation.
deletepict	The word deletepict, followed by a symbol, deletes the named picture.
deleteregion	The word deleteregion, followed by a symbol, deletes the named region.
deletesprite	The word deletesprite, followed by a symbol, deletes the named sprite.
drawpict	The word drawpict, followed by a symbol, draws the named picture. Optionally there may follow four numbers specifying a destination rectangle in which the picture is scaled and drawn, and source rectangle that specifies the area of the picture to use in the operation. These rectangles are specified as left, top, width, and height values in pixels. The destination rectangle is relative to the top left corner of the **lcd** display area. The source rectangle is relative to the top, left corner of the picture. If not present, these rectangles are both set to be the same size as the picture.
drawsprite	The word drawsprite, followed by a symbol, draws the named sprite. Optionally this may be followed by a pair of numbers that specify a horizontal and vertical offset for drawing the sprite.
enablesprites	enablesprites 1 turns on the drawing of sprites. The message enablesprites 0 turns this feature off (the default). When sprites are enabled, **lcd** consumes more memory.
font	The word font, followed by two numbers, specifies a font ID and a font size to be used when drawing text in response to a write or ascii message. Note that most font ID numbers depend on what fonts are present in the Fonts folder in the System Folder, so the effect of a font message may vary from one computer to another.

Fonts can alternately be specified by substituting a font name instead of a font ID.

framearc	Same as paintarc except that only the unfilled outline of the arc is drawn.
frameoval	Same as paintoval except that only the unfilled outline of the oval is drawn.
framepoly	Same as paintpoly except that only the unfilled outline of the polygon is drawn.
framerect	Same as paintrect except that only the unfilled outline of the rectangle is drawn.
framergn	Same as the paintrgn message except that only the unfilled outline of the region is drawn.
frameroundrect	Same as paintroundrect except that only the unfilled outline of the rounded rectangle is drawn.
frgb	The word frgb, followed by three numbers between 0 and 255, specify an RGB value sets the current foreground color of the **lcd** object.
frontsprite	The word frontsprite, followed by a symbol, sets the named sprite's drawing order so that it is drawn last (and displayed first). This command can be used to alter the order in which sprites are drawn. (Normally, sprites are drawn in the order they are recorded.)
getpenloc	The word getpenloc outputs a message consisting of the word penloc followed by two numbers, out the **lcd** object's right outlet. The numbers represent local coordinates relative to the top-left corner of the **lcd** display area. The first number is the number of pixels to the right of that corner, and the second number is the number of pixels down from that corner.
getpixel	The word getpixel, followed by two numbers which specify the location of a pixel in local coordinates relative to the top-left corner of the **lcd** display area, outputs a message consisting of the word pixel followed by five numbers out the **lcd** object's right outlet. The first three numbers, in the range 0-255 represent the RGB values of the pixel at the specified location, followed by two numbers which specify the relative x and y coordinates of the selected pixel. If a pixel is out of range, the getpixel message will output pixel 0 0 0 x y w, where *x* and *y* are the out of range location specified.
hidesprite	Turns off the drawing of a named sprite in **lcd**.
idle	idle 1 turns on the reporting of idle mouse position over an **lcd** object. The coordinates of the mouse position are sent out the middle outlet as a two-item list as the mouse moves. The numbers represent local coordinates relative to the top-left corner of the **lcd** display area. The first number is the number of pixels to the right of that corner, and the second number is the number of pixels down from that corner. idle 0 turns off this feature, which is off by default.

line	The word line, followed by two int arguments for horizontal and vertical offset, in pixels, relative to the current pen position, draws a line from the current pen position to a point determined by the specified offset, and that point becomes the new pen position. Positive arguments draw the line to the right or down; negative arguments draw up or to the left.
linesegment	The word linesegment, followed by four int arguments that specify the endpoints of a line segment, draw a line. The numbers represent the horizontal and vertical offset of the beginning endpoint, and the horizontal and vertical offset of the finishing endpoint, in pixels, relative to the top left corner of the **lcd** display area. Optionally, a color may follow. If there is one additional int argument, the color specifies a color from Max's color palette in the same way as the color message. If there are three additional int arguments, the color specifies a color as an RGB value in the same way as the frgb message.
lineto	The word lineto, followed by two int arguments for horizontal and vertical ending point, draws a line from the current pen position to the position specified by the arguments.
local	local 0 turns off drawing in the **lcd** with the mouse; local 1 turns the feature back on. In either case, **lcd** will still report the location of the mouse as it is dragged within the object's rectangle.
move	Moves the pen position a certain number of pixels down from, and to the right of, its current position. The word move must be followed by two int arguments for horizontal and vertical offset, in pixels, relative to the current pen position. Negative arguments may be used to move the pen position up or to the left.
moveto	Sets the pen position at which the next graphic instruction will be drawn. The moveto message must include two int arguments for horizontal and vertical offset, in pixels, relative to the upper left corner of the **lcd** display area.
noclip	Removes any clipping area that may be in place.
onscreen	onscreen 1 turns on the memory-saving feature of using the onscreen window for drawing. A message of onscreen 0 turns this feature off. Onscreen mode is off by default. When not using onscreen mode, **lcd** consumes more memory, but remembers its contents so that it is not erased when covered as happens with the onscreen mode.
oprgb	The word oprgb, followed by three numbers between 0 and 255, specify an RGB value used as the opcolor for penmodes that support it. For more information on on the effects of each drawing mode, refer to the Apple Developer website at *http://developer.apple.com/documentation/QuickTime/INMAC/MACWIN/imClrQuickDraw.a.htm*

paintarc	The word paintarc, followed by six int arguments that specify the left, top, right, and bottom extremities of an oval across which the arc will be drawn, and the start and end angle in degrees, paints an arc. The extremities are specified in pixels, relative to the top left corner of the **lcd** display area. Optionally, a color may follow. If there is one additional int argument, the color specifies a color from Max's color palette in the same way as the color message. If there are three additional int arguments, the color specifies a color as an RGB value in the same way as the frgb message.
paintoval	The word paintoval, followed by four int arguments specifying the left, top, right, and bottom extremities of an oval, paints an oval. These extremities are specified in pixels, relative to the top left corner of the **lcd** display area. Optionally, a color may follow. If there is one additional int argument, the color specifies a color from Max's color palette in the same way as the color message. If there are three additional int arguments, the color specifies a color as an RGB value in the same way as the frgb message.
paintpoly	The word paintpoly may be followed by as many as 254 int arguments that would specify a series of x/y pairs that define a polygon to be painted in **lcd**. These x/y pairs are specified in pixels, relative to the top left corner of the **lcd** display area. Optionally, a color may follow the last x/y pair that is the same as the first one. If there is one additional int argument, the color specifies a color from Max's color palette in the same way as the color message. If there are three additional int arguments, the color specifies a color as an RGB value in the same way as the frgb message.
paintrect	The word paintrect, followed by four int arguments specifying the left, top, right, and bottom positions of a rectangle, paints a rectangle. The edge positions are specified in pixels, relative to the top left corner of the **lcd** display area. Optionally, a color may follow. If there is one additional int argument, the color specifies a color from Max's color palette in the same way as the color message. If there are three additional int arguments, the color specifies a color as an RGB value in the same way as the frgb message.
paintrgn	The word paintrgn, followed by a symbol, paints the named region (filled). Optionally this may be followed by a pair of integer arguments which specify a horizontal and vertical offset to which the region's coordinates will be relative, and a color. If there is one additional int argument for the color, the color specifies a color from Max's color palette in the same way as the color message. If there are three additional int arguments, the color specifies a color as an RGB value in the same way as the frgb message.
paintroundrect	The word paintroundrect, followed by six int arguments specifying the left, top, right, and bottom positions of a rectangle and the amount of horizontal and vertical roundness in pixels, paints a rounded rectangle. The edge positions are specified in pixels, relative to the top left corner of the **lcd** display area. Optionally, a color may follow. If there is one additional int argument, the color specifies a color

from Max's color palette in the same way as the color message. If there are three additional int arguments, the color specifies a color as an RGB value in the same way as the frgb message.

penmode The word penmode, followed by a number in the range 0-7, sets the transfer mode for subsequent drawing operations. The following are transfer mode constants;

Copy	0
Or	1
Xor	2
Bic	3
NotCopy	4
NotOr	5
NotXor	6
NotBic	7

For more information on the effects of each drawing mode, refer to the Apple Developer website at

http://developer.apple.com/documentation/QuickTime/INMAC/MACWIN/imClrQuickDraw.a.htm

pensize The word pensize must be followed by an int argument to set the current pensize in pixels.

readpict The word readpict followed by a symbol which specifies a filename, looks for a QuickTime graphic file (a .pct file openable on Windows using the QuickTime Picture Viewer for Windows) with that name in Max's file search path, and reads the picture file from disk into RAM. This named picture can then be drawn in **lcd** with the drawpict and tilepict messages. In response to the readpict message, the object sends a message out the right outlet of the **lcd** object consisting of the word pict followed by a symbol which specifies the name of the picture file and two numbers which specify the file's width and height. If the read is unsuccessful, the error message pict <pictname> error will be sent out the right outlet.

recordregion Initiates the recording of drawing commands which will be stored in a named region. While recording, drawing commands will have no visible effect on the contents of the **lcd** object's window.

recordsprite Initiates the recording of drawing commands which will be stored in a named sprite. While recording, drawing commands will have no effect on the contents of the **lcd** object's window.

reset Erases the contents of **lcd** and resets pen state to default values. The reset message is equivalent to the sequence

clear

	pensize 1 penmode 0 frgb 0 0 0(black) brgb 255 255 255(white) moveto 0 0
scrollrect	The word scrollrect, followed by six int arguments that specify the left, top, right, and bottom positions of a rectangle to be scrolled and the number of pixels to scroll in the x and y direction, scrolls a rectangle within the **lcd** object's display area.
size	Changes the size of the **lcd** object. The word size must be followed by two int arguments which specify the dimensions (horizontal and vertical) in pixels of the new size.
textface	The word textface, followed by one or more names specifying text style(s), sets the font style(s) to be used when rendering text. Text style names are normal, bold, italic, underline, outline, shadow, condense, and extend.
textmode	The word textmode, followed by a number in the range 0-7, sets the transfer mode for subsequent drawing operations. For more information on the effects of each drawing mode, refer to the Apple Developer website at *http://developer.apple.com/documentation/QuickTime/INMAC/MACWIN/imClrQuickDraw.a.htm*
tilepict	The word tilepict, followed by a picture name argument, fills a rectangle by tiling a picture. Optionally there may follow, four numbers that specify a destination rectangle in which the picture is tiled and four numbers that specify a source rectangle that specifies the area of the picture to use in the operation. These rectangles are specified as left, top, width, and height values in pixels. The destination rectangle is relative to the top left corner of the **lcd** display area. The source rectangle is relative to the top, left corner of the picture. If not present, the destination rectangle is set to the same size of **lcd**, and the source rectangle is set to be the same size as the picture.
write	The word write, followed by any symbol, writes that symbol beginning at the current pen position, and moves the pen position to the end of the text.
writepict	The word writepict, followed by an optional filename argument, writes the current contents of the **lcd** display area to a PICT file (a .pct file openable on Windows using the QuickTime Picture Viewer for Windows). If no filename argument is present, a Save As dialog will prompt you to choose a filename and location to write the PICT file.

Inspector

The behavior of an **lcd** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **lcd** object displays the **lcd** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The size of the **lcd** display, in pixels, can be set by typing in the *Width* and *Height* number boxes. The default size of the **lcd** object is 128 pixels high and 128 pixels wide.

Checking *Local Mousing Mode* lets you draw in the **lcd** display ares with the mouse. This feature is enabled by default.

The *Draw Border* checkbox is enabled by default. Checking it creates a border around the **lcd** object's display area.

Checking the *Respond to Idle Mousing* option will report idle-time mouse positions over the **lcd** object. This feature is disabled by default.

Checking the *Onscreen Mode* option will set the **lcd** object to remembers its contents so that it is not erased when it is covered. This feature is disabled by default.

Checking the *Enable Sprites* option will enable the drawing of sprites. This feature is disabled by default. When sprites are enabled, **lcd** consumes more memory.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

Output

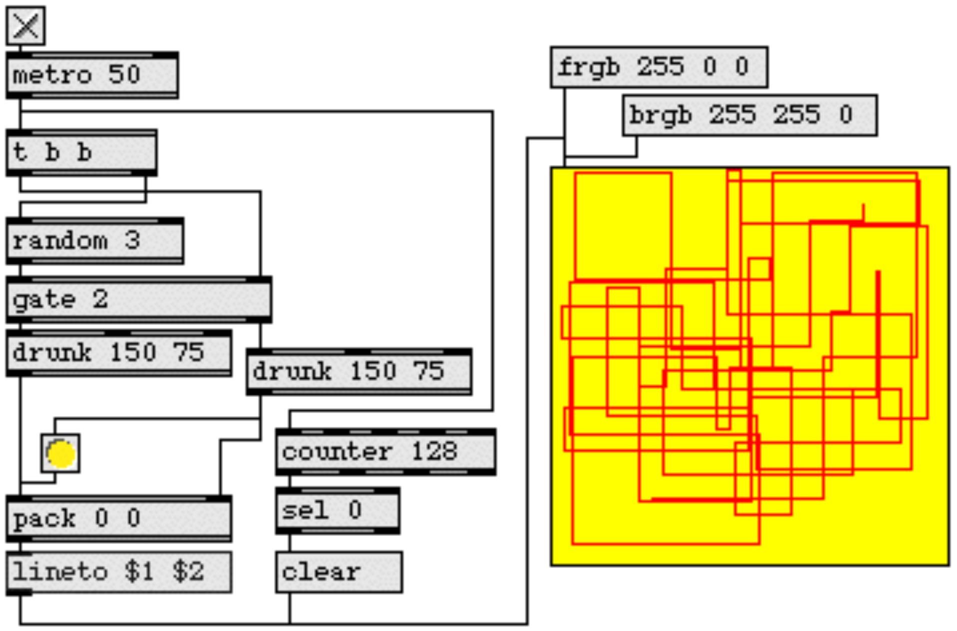
list Out 1st outlet: When you click and drag in the **lcd** display area with the mouse button held down, the coordinates of the mouse position are sent out the outlet as a two-item list as the mouse moves. The numbers represent local coordinates relative to the top-left corner of the **lcd** display area. The first number is the number of pixels to the right of that corner, and the second number is the number of pixels down from that corner.

int Out 3rd outlet: A 1 is sent out the 2nd outlet if the mouse button is currently being held down. A 0 is sent, otherwise.

list	Out 2nd outlet: When you click and drag in the **lcd** display area with the mouse button held down, the coordinates of the mouse position are sent out the outlet as a two-item list as the mouse moves. The numbers represent local coordinates relative to the top-left corner of the **lcd** display area. The first number is the number of pixels to the right of that corner, and the second number is the number of pixels down from that corner.
list	Out 1st outlet: When you draw in the **lcd** with the mouse button held down, the coordinates of the mouse position are sent out the outlet as a two-item list as the mouse moves. The numbers represent local coordinates relative to the top-left corner of **lcd**. The first number is the number of pixels to the right of that corner, and the second number is the number of pixels down from that corner.
list	Out 4th outlet: When mouse idle mode is using the idle message or by enabling the *Respond to Idle Mousing* Inspector option, a list of current mouse coordinates is sent out the third outlet when the mouse is positioned over the **lcd** object's display area.
update	Out 4th outlet: The word update is output whenever lcd receives an update message from Max telling it to redraw itself. This is only done when **lcd** is in onscreen mode
penloc	Out 4th outlet: In response to the getpenloc message, **lcd** outputs a message consisting of the word penloc followed by two numbers representing the pen location in local coordinates relative to the top-left corner of the **lcd** display area. The first number is the number of pixels to the right of that corner, and the second number is the number of pixels down from that corner.

lcd

Examples



Draw an angular snake diagram using **lcd**

See Also

frame	Draw framed rectangle in a graphic window
graphic	Window for drawing sprite-based graphics
mousestate	Report the status and location of the mouse
oval	Draw solid oval in a graphic window
panel	Colored background area
rect	Draw solid rectangle in a graphic window
ring	Draw framed oval in a graphic window
Tutorial 43	Graphics in a patcher
Graphics	Overview of Max graphics windows and objects

led

Display on/off status in color

Input

int	If the number is 0, **led** shows its darkened state, and outputs 0. If the number is not 0, **led** shows its brightened state and outputs 1.
float	Converted to int.
bang	Flashes **led** on and off quickly, and outputs 0.
	Clicking on an **led** toggles it back and forth between bright and dark, outputting 1 and 0.
blinktime	In left inlet: the word blinktime, followed by a number, specifies the duration (in milliseconds) that **led** will flash when it is clicked upon or receives a bang message.
pict	In left inlet: the word pict, followed by an integer from 0 to 4, changes the color used by **led**.
set	The word set, followed by a non-zero number causes **led** to show its brightened state, but causes no output; set 0 shows the **led** object in a darkened state, but causes no output.
toggle	Switches the **led** from dark to bright and sends 1 out the outlet; or vice-versa, from bright to dark, sending 0 out the outlet.

Inspector

The behavior of an **led** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **led** object displays the **led** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **led** Inspector lets you set the following attributes:

The *LED Pict* option lets you use from among five colors for the **led** object's display: red (the default), green, blue, yellow, or black and white.

Flash Time specifies the duration (in milliseconds) that **led** will flash when it is clicked upon or receives a bang message. The default is 150.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

led

Display on/off status in color

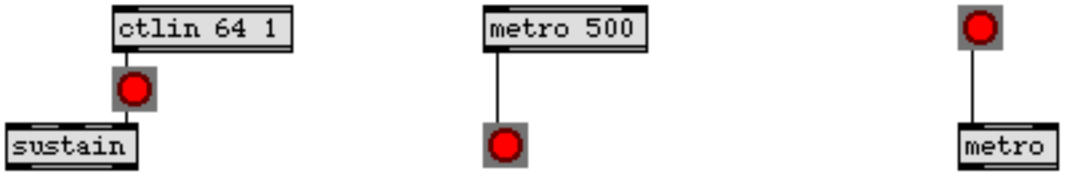
Arguments

None.

Output

int	The output is 1 when **led** is bright, 0 when it is dark. A bang in the inlet flashes **led** on and off and sends 0 out the outlet.

Examples



Displays an on/off state, announces activity with a flash, or can be used as a toggle

See Also

button	Flash on any message, send a bang
pictctrl	Picture-based control
togedge	Report a change in zero/non-zero values
toggle	Switch between on and off (1 and 0)
Tutorial 40	Automatic actions

line

Output numbers in a ramp from one value to another

Input

list — The first number specifies a *target value*, and the second number specifies a total amount of time (in milliseconds). In that amount of time, numbers are output regularly in a line from the currently stored value to the target value.

int or float — In left inlet: The number is the target value, to be arrived at in the time specified by the number in the middle inlet. If no time has been specified since the last target value, the time is considered 0 and **line** immediately outputs the target value.

Note: the output type for the **line** object is set by using the first argument to the object (see Arguments).

In middle inlet: The number is the time, in milliseconds, in which to arrive at the target value.

In right inlet: The number is the interval (in milliseconds) at which intermediary numbers are regularly sent out.

clock — The word clock, followed by the name of an existing **setclock** object, sets **line** to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets **line** back to using Max's regular millisecond clock.

stop — In left inlet: Stops **line** from sending out numbers, until a new target value is received.

set — In left inlet: The word set, followed by a number, makes that number the new starting value from which to proceed to the next received target value. The set message also stops **line** if it is in the process of sending out numbers.

Arguments

int or float — Optional. The first argument sets the output type for the object—if the first argument is an int, the line object outputs integer values, and a float will set the line object to output floating point values. The first argument also sets the initial value to be stored in **line** and the output type for the object. If there is no argument, the initial value is 0 and the output type is int. The second argument sets an initial value for the *grain*, the time interval at which numbers are sent out. If the grain is not specified, **line** outputs a number every 20 milliseconds. The minimum grain allowed is 1 millisecond; any number less than 1 will be set to 20.

Output

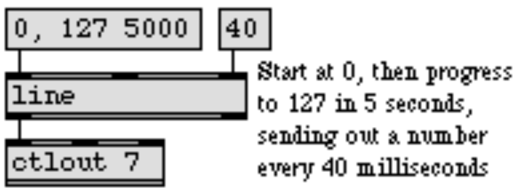
int — Out left outlet: Numbers are sent out at regular intervals, describing a straight line toward a target value. If a new target value and time are specified before the line is completed, the new line starts from the most recent output value, in order to avoid discontinuities.

If a value is received in the left inlet without an accompanying time value, it is sent out immediately (time is considered 0).

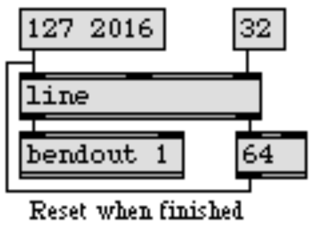
bang Out right outlet: When **line** has arrived at its target value, bang is sent out.

Note: In practice, the target value is arrived at in just under the amount of time specified (time minus grain).

Examples


0, 127 5000
40
line
ctlout 7
Start at 0, then progress to 127 in 5 seconds, sending out a number every 40 milliseconds

Output values in a straight line...



and bang *when finished*

See Also

envi	Script-configurable envelope in a patcher window
funbuff	Store x,y pairs of numbers together
setclock	Control the clock speed of timing objects remotely
uzi	Send a specific number of bang messages
Tutorial 31	Using timers

loadbang

Send a bang *automatically when patch is loaded*

Input

There are no inlets. Output is triggered automatically when the file is opened, or when the patch is part of another file that is opened.

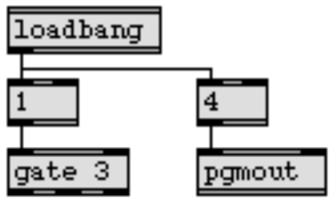
Arguments

None.

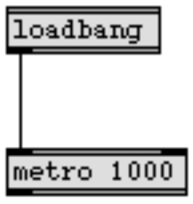
Output

bang	Sent automatically when the patch is loaded. You can also cause **loadbang** to send out a bang by double-clicking on it in a locked patcher, or by sending a loadbang message to a **thispatcher** object in the same patcher. Holding down the Shift and Command keys on Macintosh or Shift and Control keys on Windows while a patch is loading prevents **loadbang** objects in that patch from sending any output.

Examples



Set initial values when a patch is loaded...



or start a process automatically

See Also

active	Send 1 when patcher window is active, 0 when inactive
button	Flash on any message, send a bang
closebang	Send a bang when patcher window is closed
thispatcher	Send messages to a patcher
Tutorial 40	Automatic actions

makenote

Generate a note-off message following each note-on

Input

int — In left inlet: The number is treated as a pitch value for a MIDI note-on message. It is paired with a velocity value and the numbers are sent out the outlets. After a certain time, a note-off message (a note-on with a velocity of 0) is sent out for that pitch.

In middle inlet: The number is stored as a velocity to be paired with pitch numbers received in the left inlet.

In right inlet: The number is stored as the duration (in milliseconds) that **makenote** waits before a note-off message is sent out.

float — Converted to int.

list — The second number is treated as the velocity and is sent out the right outlet. The first number is treated as the pitch and is sent out the left outlet. A corresponding note-off message is sent out later.

stop — Causes **makenote** to send out immediate note-offs for all pitches it currently holds.

clear — Erases all notes currently held by **makenote**, *without* sending note-offs.

Arguments

int — Optional. The first argument sets an initial velocity value to be paired with incoming pitch numbers. If there is no argument, the initial velocity is 0.

The second optional argument sets an initial note duration (time before a note-off is sent out), in milliseconds. If the second argument is not present, the note-off follows the note-on immediately.

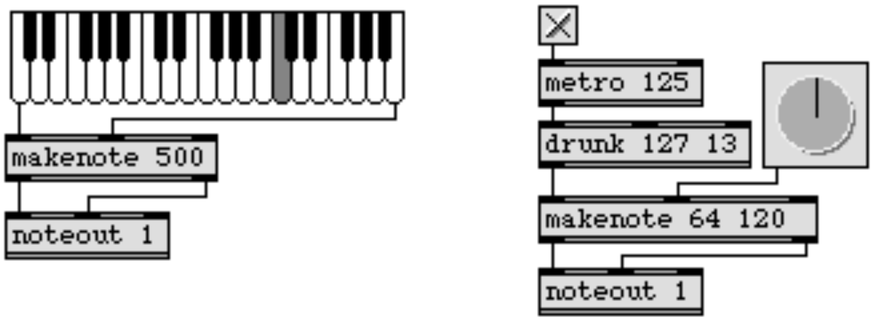
float — Converted to int.

Output

int — Out left outlet: The number received in the left inlet is sent out immediately, paired with a velocity value out the other outlet. After a certain duration, the same number is sent out paired with a velocity of 0.

Out right outlet: The number in the middle inlet is sent out as a velocity value in conjunction with a pitch value out the left outlet. After a certain duration, 0 is sent out paired with the same pitch.

Examples



Supply note-offs for note-ons generated within Max

See Also

flush	Provide note-offs for held notes
midiout	Transmit raw MIDI data
noteout	Transmit MIDI note messages
stripnote	Filter out note-off messages, pass only note-on messages
xnoteout	Format MIDI note messages with release velocity
Tutorial 13	Managing note data

match

Look for a series of numbers output it as a list

Input

int	If the numbers match the arguments, in the proper order, they are sent out as a list.
clear	Causes **match** to forget all numbers it has received up to that time.
set	The word set, followed by a list of numbers, specifies a new series of numbers **match** will look for.

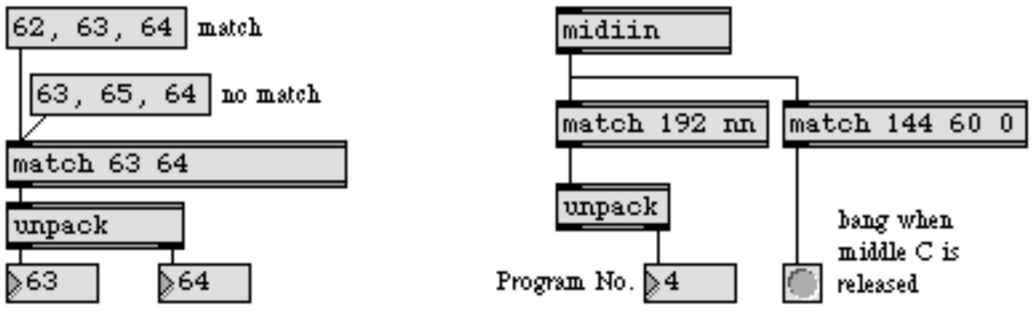
Arguments

list	Obligatory. The arguments specify numbers to look for, in the proper order. The word nn can be used as a *wild card* that will match any number.

Output

list	The numbers received in the inlet are compared with the arguments. If the numbers are the same, and in the same order, they are sent out the outlet as a list.

Examples



Numbers must be the same, and in the same order

See Also

iter	Break a list up into a series of numbers
pack	Combine numbers and symbols into a list
select	Select certain inputs, pass the rest on

matrixctrl

Matrix switch control

matrixctrl is a user interface object that consists of a rectangular grid of switch-like controls called *cells*. All of the cells in a **matrixctrl** object have the same appearance and behavior. Each cell has two or more states. By default, the cells have two states, representing "off" and "on." You can create cells with any number of states. Clicking on a cell increases its state by one. After a cell reaches its last state, it returns to its zero state when clicked again—thus, a cell with only two states will toggle back and forth between these states with each mouse click.

matrixctrl was originally constructed to control the MSP object **matrix~**, but is useful for other user interface applications, such as groups of switches, groups of visual indicators, and drum-machine-oriented sequencers.

Note: The **matrixctrl** object requires that QuickTime be installed on your system to open any files other than PICT files (i.e., files with a .pct extension on Windows). If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.

Input

(Mouse) A mouse click on a cell will increase its value by one. Values in **matrixctrl** will wrap back to 0 once they have reached their maximum possible state. Dragging across several cells will set their values to that of the first cell clicked. Dragging across cells while holding down the Shift key will allow you to drag in straight horizontal or vertical lines only.

bang A bang causes **matrixctrl** to dump its current state in lists of three values for each cell pair, in the format

horizontal-coordinate vertical-coordinate value

list A list of ints sets cells in the **matrixctrl** object using the format <horizontal-coordinate vertical-coordinate value>. Multiple triplets of values can be used to set more than one cell. Coordinates for the cells start at 0 in the upper-left hand corner and the values for each cell start at 0 and go up to the value range minus one, set by the object's inspector. Substituting the symbols inc and dec in place of the value will increment or decrement that cell coordinate by a value of one. Changing the cell state with a list causes the list to be output from **matrixctrl**.

set The word set, followed by a list as described above, changes the state of **matrixctrl** without echoing the values to the output.

active The word active, followed by a 0 or 1, causes **matrixctrl** to ignore or respond to mouse clicks, respectively. By default, **matrixctrl** responds to mouse clicks.

bkgndpicture The word bkgndpicture, followed by a symbol that specifies a filename, designates the graphics file that the **matrixctrl** object will use for the matrix background image. The **matrixctrl** object accepts PICT files and, if QuickTime Version 3.0 or later is installed, other picture file formats that are listed in the QuickTime appendix. The symbol used as a filename must either be the name of a file in Max's cur-

	rent search path, or an absolute pathname for the file (e.g. "MyDisk:/Documents/UI Pictures/CoolBkgnd.pct"). The word bkgndpicture by itself puts up a standard Open Document dialog box and displays the common graphics files supported by QuickTime.
cellpicture	The word cellpicture, followed by a symbol that specifies a filename, designates the graphics file that the **matrixctrl** object will use for each cell. The **matrixctrl** object accepts PICT files and, if QuickTime Version 3.0 or later is installed, other picture file formats that are listed in the QuickTime appendix.The symbol used as a filename must either be the name of a file in Max's current search path, or an absolute pathname for the file (e.g. "MyDisk:/Documents/UI Pictures/Cell.pct"). The word cellpicture by itself puts up a standard Open Document dialog box and displays the common graphics files supported by QuickTime.
clickedimage	The word clickedimage, followed by a nonzero value, specifies that the graphics file used by the **matrixctrl** object contains an additional image to be displayed when a cell is clicked.
clickvalue	The word clickvalue, followed by a number, toggles the click value mode. If the clickvalue message is followed by a 0 or a positive number, clicking on a cell sets its value to the given number. If clickvalue is followed by a negative number, the matrixctrl object reverts to its default behavior in which clicking a cell increments its value. The clickvalue message allows the use of the **matrixctrl** object to create grid editors by creating graphics files which contain a sequence of images, each of which is assigned to a different value; as you click through the sequence of images, the cell image will change to reflect velocity, note, etc.
disablecell	The word disablecell, followed by a list of number pairs which specify the horizontal and vertical coordinates of a cell or cells, sets the designated cell or cells so that they do not respond to mouse clicks. The disablecell message expects at least one pair of numbers, but more may be added to disable multiple cells (e.g., disable 0 0 3 4 9 12). Although disabled cells will ignore mouse clicks, their values can be set using messages.
enablecell	The word enablecell, followed by a list of number pairs which specify the horizontal and vertical coordinates of a cell or cells, will set any designated cell or cells which have been disabled using the disablecell message to respond to mouse clicks again. The enablecell message expects at least one pair of numbers, but more may be added to enable multiple cells (e.g., enable 1 1 1 2 2 2).
getrow	The word getrow, followed by a number, sends the values of the cells in the row designated by the number out its right outlet.
getcolumn	The word getcolumn, followed by a number, sends the values of the cells in the column designated by the number out its right outlet.

horizontalmargin	The word horizontalmargin, followed by a number, sets a horizontal margin (in pixels) between the outermost cells and the edge of the matrixctrl object's bounding box.
horizontalspacing	The word horizontalspacing, followed by a number, sets the horizontal distance (in pixels) between adjacent cells in the matrixctrl object.
imagemask	The word imagemask, followed by a nonzero value, specifies that the **matrixctrl** cell graphics file has additional rows of images for use as image masks.
inactiveimage	The word inactiveimage, followed by a nonzero value, specifies that the **matrixctrl** cell graphics file has additional rows of images for use in an inactive state (set with an active 0 message).
invisiblebkgnd	The word invisiblebkgnd, followed by a nonzero value, specifies that the **matrixctrl** will be drawn without a background image, and its cells will be superimposed over any underlying Max objects. invisiblebkgnd 0 disables this feature.
one/row	The word one/row, followed by a nonzero value, only allows one cell per row to have a non-zero state. Setting any cell in a row to a non-zero state causes any other non-zero cells to change to the zero state. one/row 0 removes this constraint.
one/column	The word one/column, followed by a nonzero value, only allows one cell per column to have a non-zero state. Setting any cell in a column to a non-zero state causes any other non-zero cells to change to the zero state. one/column 0 removes this constraint.
one/matrix	The word one/matrix, followed by a nonzero value, only allows one cell in the entire object to have a non-zero state. Setting any other cell in the matrix to a non-zero state causes any other non-zero cells to change to the zero state. one/matrix 0 removes this constraint.
range	The word range, followed by an int, sets the number of possible states each cell can have. It must be set to a value of at least 2 (for states 0 and 1).
verticalmargin	The word verticalmargin, followed by a number, sets a vertical margin (in pixels) between the outermost cells and the edge of the matrixctrl object's bounding box.
verticalspacing	The word verticalspacing, followed by a number, sets the vertical distance (in pixels) between adjacent cells in the **matrixctrl** object.

Inspector

The behavior of a **matrixctrl** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **matrixctrl** object displays the

matrixctrl Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The *Cell Spacing* number boxes set the horizontal and vertical distance (in pixels) between adjacent cells in the **matrixctrl** object.

The *Margin* number boxes are used to specify horizontal and vertical margins (in pixels) between the outermost cells and the edge of the object's bounding box.

Checking the *Has Clicked Images* option will use an alternate set of image frames in your graphics file to give the cell a different appearance when the user clicks and drags it.

The *Has Inactive Images* checkbox tells the **matrixctrl** object that your graphics files have additional images for the cell's inactive state. Leave this box unchecked if the picture files used by the control do not have these images.

If you want to use image masks in your cell's graphics file to draw the cell, select the *Has Image Mask* option. Masks can be used to create cells with a non-rectangular shape. If your cell picture has separate images for the clicked and/or inactive state, you must supply masks for those as well.

Checking the *Invisible Background* box tells the **matrixctrl** object not to draw anything for the background of the matrix. The cells will appear to "float" over any underlying objects.

The *One Per Column*, *One Per Row*, and *One Per Matrix* checkboxes define the **matrixctrl** object's behavior. If checked, **matrixctrl** only allows one cell per column, row, or in the entire object to have a non-zero state. Setting any cell to a non-zero state causes any other non-zero cells to change to the zero state.

Cell Value Range is used to set the number of possible states each cell can have. It must be set to a value of at least 2 (for states 0 and 1).

Cell Picture File and *Background Picture File* lets you choose graphics files for the matrix cells and its background by clicking on the Open buttons. It can open PICT files and, if QuickTime Version 3.0 or later is installed, other picture file formats that are listed in the QuickTime appendix. The current file's name appears in the text box to the left each of the buttons. You can also choose a file by typing its name in this box, or by dragging the file's icon from the Finder into this box.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

matrixctrl

Matrix switch control

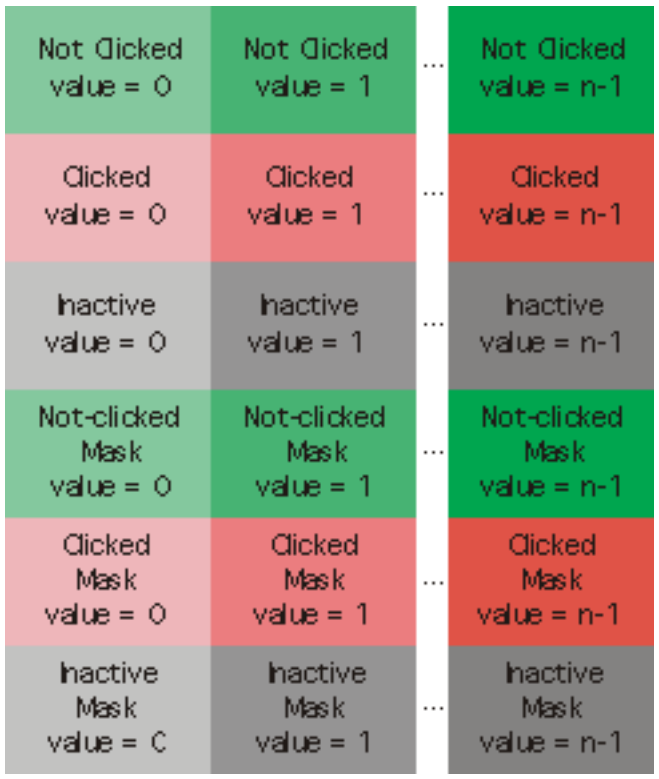
Arguments

None.

Picture File Format

Background picture files for **matrixctrl** can be any Macintosh PICT file or, if QuickTime Version 3.0 or later is installed, other picture file formats that are listed in the QuickTime appendix.If the **matrixctrl** is larger than the chosen picture, copies of the picture will be added to fill the object.

Cell picture files must be in the following format:



The picture is made up of a grid of images. All images have the same width and height. Each column of images represents one cell state. The picture must have at least two columns, since cells must have at least two states.

The first row of images is used for the idle (or "not clicked") appearance of the cells. The first row of images is mandatory; all subsequent rows are optional. The second row are images for the clicked appearance; these images will be used to draw the cell when it is clicked. The appearance of the cell reverts to its idle image when the mouse is released. The third row of images are used when the **matrixctrl** is in its inactive state, i.e. when it has received an active 0 message.

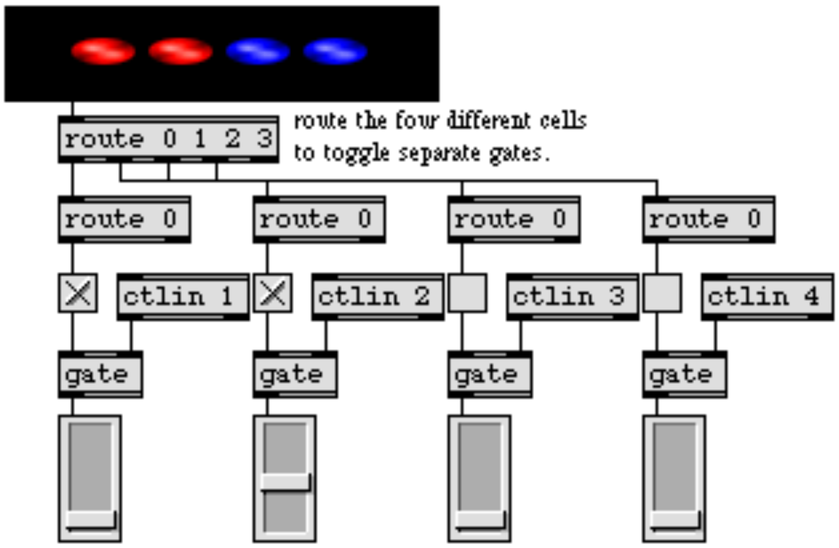
Image masks can be used to create cells with non-rectangular outlines. These masks are in the lower rows of the picture file. If you wish to use masks for any of the cell images, you must provide masks for all of them—each row of images will have a corresponding row of masks. Like all masks for Max's picture-based controls, black pixels create areas of the corresponding image that will be drawn, and while pixels create invisible areas.

Output

list When a cell changes state in response to a mouse click, a list is sent out the **matrixctrl** object's left outlet. The list contains the row, column, and value (state) of the clicked control. Individual cells can also be set by sending lists to the object's left inlet. Rows and columns are numbered starting with zero, at the upper-left corner of the matrix.

The numbers received in the inlet are compared with the arguments. If the numbers are the same, and in the same order, they are sent out the outlet as a list.

Examples



matrixctrl *can be used to control multiple gates and switches at once*

See Also

dial	Output numbers by moving a dial onscreen
hslider	Output numbers by moving a slider onscreen
kslider	Output numbers from a keyboard onscreen
pictctrl	Picture-based control
pictslider	Picture-based slider
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
ubutton	Transparent button, sends a bang
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

maximum

Output the greatest in a list of numbers

Input

int	In left inlet: If the number is greater than the value currently stored in **maximum**, it is sent out the outlet. Otherwise, the stored value is sent out. In right inlet: The number is stored for comparison with subsequent numbers received in the left inlet.
float	Converted to int, unless there is a float argument, in which case all numbers are compared as floats.
list	In left inlet: The numbers in the list are all compared to each other, and the greatest value is sent out the outlet. The value stored in **maximum** is replaced by the *next greatest* value in the list.The **maximum** object accepts lists of up to 256 elements.
bang	In left inlet: Sends the most recent output out the outlet again.

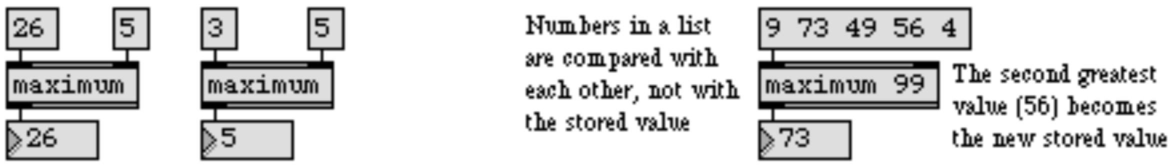
Arguments

int or float	Optional. Sets an initial value to be compared with numbers received in the left inlet. If the argument contains a decimal point, all numbers are compared as floats, and the output is a float. If there is no argument, the initial value is 0.

Output

int	The number received in the left inlet is compared with the value currently held by **maximum** (or numbers received as a list are compared with each other), and the greatest of the numbers is sent out the outlet.
float	Only if there is an argument with a decimal point.

Examples


26
5
maximum
26
3
5
maximum
5
Numbers in a list are compared with each other, not with the stored value
9 73 49 56 4
maximum 99
73
The second greatest value (56) becomes the new stored value

The output is the greater of two numbers, or the greatest in a list of numbers

See Also

minimum	Output the smallest in a list of numbers
past	Report when input increases beyond a certain number
peak	If a number is greater than previous numbers, output it
>	*Is greater than*, comparison of two numbers

mean

Find the running average of a stream of numbers

Input

int or float	The number is added to the sum of all numbers received up to that point, and the mean is sent out.
bang	Sends out the previous output (the stored average value).
list	The numbers in the list are added together, the sum is divided by the number of items in the list, and the mean is sent out. All previously received numbers are cleared from memory.
clear	Resets the contents of the object to zero.

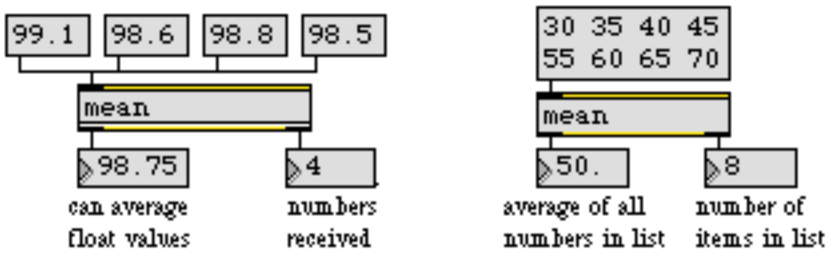
Arguments

None.

Output

float	Out left outlet: The mean (average) value of all numbers received up to that point, or of all the numbers received together in a list.
int	Out right outlet: How many numbers have been included in the averaging process.

Examples



Find the average value of many numbers

See Also

accum	Store, add to, and multiply a number
anal	Make a histogram of number pairs received
bag	Store a collection of numbers
histo	Make a histogram of the numbers received
prob	Make weighted random series of numbers

menubar

Put up a custom menu bar

The **menubar** object provides control over the Macintosh menu bar. It allows your patch to put up its own menus, and add items to standard File and Edit menus. When a menu item is chosen, the item number is sent out the outlet corresponding to the menu containing the item. You configure the **menubar** by writing a script in a text editor window available by double-clicking on the object in a locked patcher.

Input

int	A nonzero number displays the **menubar** object's menus, 0 restores the previous contents of the menu bar (either the Max menus or the menus of another **menubar** object).
checkitem	Followed by a menu number, an item number, and a code 0 or 1, checkitem puts a check before the specified item if the code is 1, otherwise it removes the check.
enableitem	Followed by a menu number, an item number, and a code 0 or 1, enableitem enables the specified item if the code is 1, otherwise it disables (and grays out) the item.
markitem	(Macintosh only) Followed by a menu number, an item number, and an ASCII character code, markitem places the character next to the specified item. Common mark character ASCII codes are 18 for the check mark and 19 for the diamond mark. You may also wish to use the em dash (209) or bullet (165).
(menu bar)	When the **menubar** object has been activated (by a nonzero number in its inlet) and an item is selected in the menu bar, the menu number and item number are received by the **menubar** object, and the item number is sent out the appropriate outlet.

Arguments

int	Optional. The first argument sets the number of menus in the object's menu bar. If present, it must be at least 5 (one additional menu). The four default menus, which are always present, are File, Edit, Windows., and Help. On Macintosh, the Standard System Menu with the Apple icon and the Max/MSP application menu will appear to the left of the other menus. The second optional argument is a numerical code to indicate that certain items in the default menus are to be removed from those menus. The code is a sum of the following values assigned to the commands to be suppressed: 1=**Overdrive** in the Options menu, 2=**Resume**, and 4=**Midi Setup**.... in the File menu For example, to eliminate the **Overdrive** and **Midi Setup** commands from the Edit menu, the appropriate second argument is 5 (1+4).

Script Messages

You define a **menubar** with a series of script messages, typed into a text editor window opened by double-clicking on a **menubar** object in a locked patcher. When you close the script window and confirm saving the changes, the script file is interpreted. If there are no errors, the customized menu bar will be ready for use when **menubar** receives a nonzero number in its inlet.

Each message should be preceded by #X and end with a semicolon (;). The first script message must be apple and the last end. An example script follows the definition of the messages.

Messages to Modify Standard Menus

Message	*Arguments*
about	• Text of the first menu item (i.e. About My Program…). On the Macintosh the About item appears as the first item in the application menu (Max/MSP menu). On Windows, it appears as the first item in the Help menu. The message apple may be used optionally for compatibility with older Macintosh versions of Max.
file	• Item number to output • Text of item to add to file menu The file message inserts items at the top of the standard File menu (before the **Midi Setup...** menu item). Each item has a number associated with it which is sent out the when the item is chosen. The order in which your additional items appear in the File menu is determined by their order in the script, not by the (arbitrary) number associated with each item.
edit	• Item number to output • Text of item to add to edit menu The edit message inserts items into the standard Edit menu after the **Clear** item and before the **Overdrive** and **Resume** items (which are moved into the Edit menu when **menubar** is activated). A blank line separates the custom inserted items from the default items. Each item has a number associated with it which is sent out the third outlet of **menubar** when the item is chosen. The order in which your additional items appear in the Edit menu is determined by their order in the script, not by the (arbitrary) number associated with each item.
newitem	• Item number to output. The newitem message followed by a non-zero number directs Max to send the specified number out the **menubar object's** File menu outlet when the user chooses the **New** command from the File menu, instead of opening a new patcher window. The message newitem 0 (or the absence of any newitem message) causes the **New** command to behave normally.

open	• Item number to output.

The open message followed by a non-zero number directs Max to send the specified number out the **menubar object's** File menu outlet when the user chooses the Open... command from the File menu, instead of displaying the Open Document dialog box. The message open 0 (or the absence of any open message) causes the Open... command to behave normally.

closeitem	(No arguments.)

Causes a Close item to appear in the File menu, for closing the active window.

saveas	• Item number to output.

The saveas message followed by a non-zero number directs Max to send the specified number out the **menubar** object's File menu outlet when the user chooses Save or Save As… from the File menu, instead of performing the standard Save actions. The number sent out the outlet when Save is chosen will be 1 less than the number sent when Save As… is chosen. The message saveas 0 (or the absence of any saveas message) causes the Save and Save As... commands to behave normally.

Messages for Creating New Menus and Items

Message	*Arguments*
menutitle	• Menu number (must be at least 5 and must not exceed the number of outlets specified in the argument to **menubar** • Name of menu

The menutitle message adds a new menu before the Window menu. The first additional menu is number 5. The menu number determines both the order of the additional menu in the menu bar and the outlet it uses when the user chooses its items. A menutitle message must appear in the script *before* any item messages that refer to its menu number.

item	• Menu number • Item number • Text of item • (Optional.) "Meta-characters"

The item message adds an item to an additional menu previously defined with a menutitle message. The order in which your items appear in the menu is determined by their order in the script, not by the (arbitrary) number associated with each item. The item number argument only specifies the number which is sent out the **menubar** object's outlet when the user chooses this item. It's a good idea to start your item numbers at 1 and list the items in the order you want them to appear in a menu.

You can alter the appearance of a menu item by including "meta-characters" in the item text. For more on metacharacters, consult the Apple QuickTime Developer documentation found at:

http://developer.apple.com/documentation/Carbon/Reference/Menu_Manager/menu_mgr_ref/function_group_4.html

A few of the recognized meta-characters are:

/ followed by a character, assigns that character as a Command-key equivalent
< followed by B, I, O, S, or U, specifies a font style (such as O for outline)
! followed by a character, marks the menu item with that character
(disables the menu item

Thus, these special characters cannot appear as part of the actual item text. For example, the text On/Off will appear as "Onff_O", not as "On/Off".

Completing the Script Definition

Message	*Arguments*
end	(No arguments.)

The end message builds the menus and reports any errors encountered.

Output

int The default **menubar** object has four outlets. If the **menubar** object has been activated (by receiving a nonzero number in its inlet), the leftmost outlet sends a 1 when the first item in the Apple menu is chosen. The second outlet sends the item number when an extra item is chosen from the File menu. The third outlet sends the item number when an extra item is chosen from the Edit menu. The fourth outlet sends an item number when the user chooses an item from the Windows menu. If additional menus have been defined, item numbers are sent out the additional outlets to the right, starting with the fifth one.

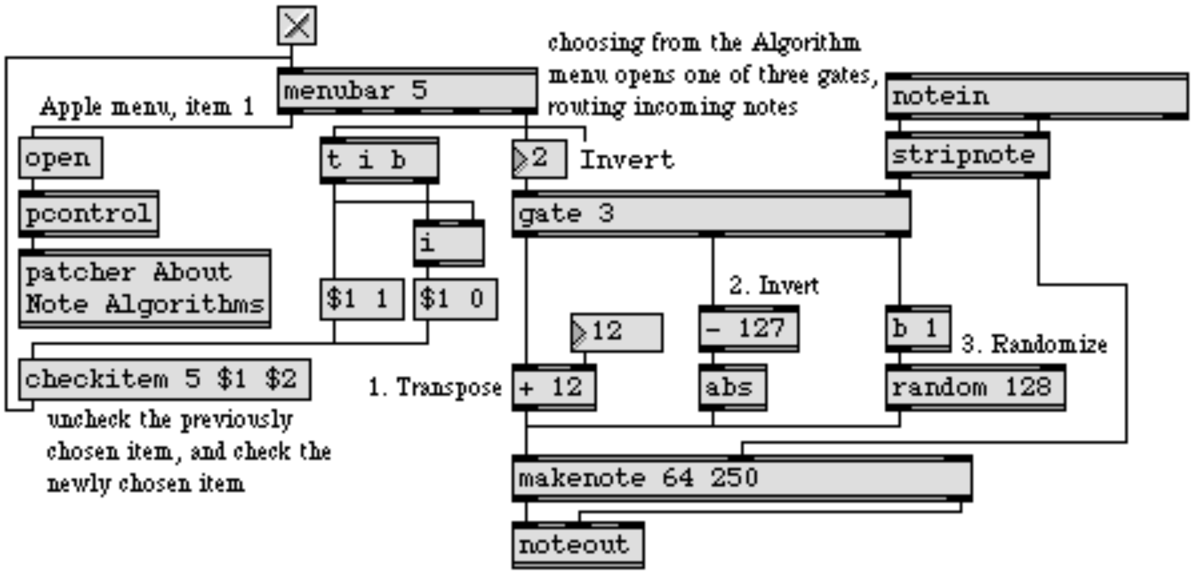
Examples

Here is an example **menubar** script:

```
#X about About Note Algorithms…;
#X closeitem;
#X menutitle 5 Algorithm;
#X item 5 1 Transpose;
#X item 5 2 Invert;
#X item 5 3 Randomize;
#X end;
```

Note that we suggest capitalizing each letter in a menu item to maintain a consistent style with other items in the menu.

The above script is used in a **menubar** in the following example, which uses the extra menu to switch among three note-processing algorithms.



An implementation of the example **menubar** *script*

See Also

umenu	Pop-up menu to display and send commands
Menus	Explanation of commands

message

Send any message

Input

The **message** object (a box that displays and sends out a message) is often referred to as the **message** *box*, in order to distinguish it from a *message* (the data that is actually sent from one object to another).

bang	Sends out the contents of the **message** box. A mouse click on the **message** box has the same effect.
int or float	The number replaces the value stored in the argument $1, if such an argument exists, then sends out the contents of the **message** box.
list	Each item in the list is stored in place of its corresponding $ argument, if such an argument exists, then the contents of the **message** box are sent out.
append	The word append, followed by a message, appends that message (preceded by a space) at the end of the contents of the **message** box, without triggering output.
color	The word color, followed by a number from 0 to 15, sets the color of the **message** box to one of the object colors which are also available via the Color command in the Object menu.
open	Opens the **message** Inspector window. If the word open is followed by a 1, the contents of the **message** box will be sent out its outlet when the text field in the Inspector window is changed or the Inspector window is closed. The second optional argument to the open message is a symbol which specifies the prompt that will appear at the top of the dialog box. The default prompt is *Set Message Text*. Use double quotes if you want to include spaces in the prompt.
prepend	The word prepend, followed by a message, places that message (followed by a space) before the beginning of the contents of the **message** box, without triggering output.
set	The word set, followed by a message, sets the contents of the **message** box to that new message, without triggering output. The word set by itself erases the contents of the **message** box.
symbol	The word symbol, followed by a symbol, stores that symbol in the $1 argument, then sends out the contents of the **message** box.

Inspector

The contents of the **message** object can be changed by selecting the object and choosing Get Info… from the Object menu. You cannot use the Inspector for the **message** object in a floating window.

Typing in the *Set Message Text* text area specifies the contents of the **message** box.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

anything — The initial contents of the **message** box are typed in when the patcher window is unlocked. Any message of up to 256 items can be contained in a **message** box. Certain characters have special meaning.

$ — A dollar sign ($), followed immediately by a number in the range 1-9, is a changeable argument. This argument's value can be replaced by the corresponding item in a list received in the inlet. (Example: $2 stores the second item in a list as its value before sending out the contents of the **message** box.) The value of a changeable argument is initially 0.

, — A comma (,) divides a message into separate messages which will be sent out in order. (Example: 3, 4, 5 sends out 3, then 4, then 5.)

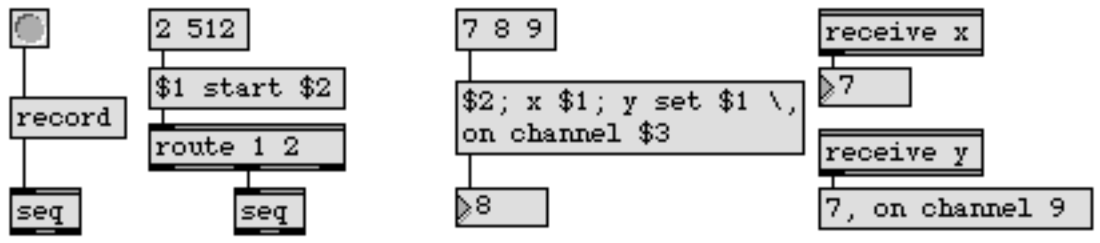
; — A semicolon (;) sends a message to a **receive** object. The first item following a semicolon is the name of the **receive** object. The rest of the message (or up to the next semicolon) is sent to that object, rather than out the outlet. The first item after the semicolon can be a changeable argument, so an incoming message can set the destination of the message "on the fly."

\ — A backslash (\) is used to negate the special traits of a special character. When a backslash immediately precedes a dollar sign, comma, or semicolon, the character is treated as a normal character. (Example: Notes played were C\, E\, and G.)

Output

anything — The contents of the **message** box are normally sent out the outlet. If a semicolon is present, the rest of the message (or up to the next semicolon) is sent to the specified **receive** object, rather than out the outlet.

Examples



Send a simple message, or construct a message of any degree of complexity

See Also

append	Append arguments at the end of a message
prepend	Place one message at the beginning of another
receive	Receive messages without patch cords
Tutorial 1	Saying “Hello!”
Tutorial 25	Managing messages

metro

Output a bang *message at regular intervals*

Input

int or float — In left inlet: Any number other than 0 starts **metro**. At regular intervals, **metro** sends a bang out the outlet. 0 stops **metro**.

In right inlet: The number is the time interval, in milliseconds, at which **metro** sends out a bang. A new number in the right inlet does not take effect until the next output is sent. The **metro** object's minimum interval time is .02 second.

bang — In left inlet: Starts **metro**.

stop — In left inlet: Stops **metro**.

clock — The word clock, followed by the name of an existing **setclock** object, sets the **metro** to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets **metro** back to using Max's regular millisecond clock.

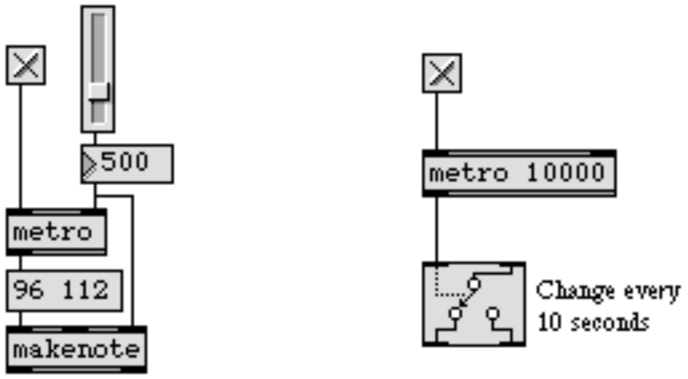
Arguments

int or float — Optional. The first argument sets an initial value for the time interval at which **metro** sends its output. If there is no argument, the initial time interval is 5 milliseconds. Any argument less than 5 will be set to 5. If the second argument is 1, **metro** uses the MIDI Manager external clock (see the ext message discussion above). If the second argument is 0 or not present, **metro** uses Max's internal millisecond clock.

Output

bang — A bang is sent immediately when **metro** is started, and at regular intervals thereafter.

Examples



Repeatedly send a message or trigger a process

See Also

clocker	Report the elapsed time, at regular intervals
counter	Count the bang messages received, output the count
delay	Delay a bang before passing it on
setclock	Control the clock speed of timing objects remotely
tempo	Output numbers at a metronomic tempo
uzi	Send a specific number of bang messages
Tutorial 4	Using **metro**

midiflush

Send note-offs for hanging note-ons in raw MIDI data

Input

int	**midiflush** expects raw MIDI data from a source such as **seq** or **midiin**. **midiflush** passes the data through unchanged, and observes which note-on messages on each channel have not received matching note-off messages.
bang	When **midiflush** receives a bang, it outputs MIDI note-off messages for all note-ons which have not been matched by note-offs since the object was created (or the last bang message was sent).
clear	Erases any note-ons held by **midiflush**, without sending any note-offs.

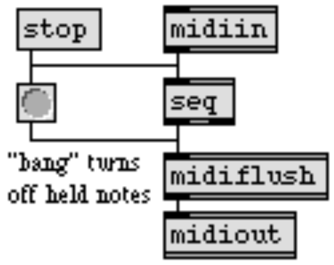
Arguments

None.

Output

int	**midiflush** passes all its input through to its output, and sends MIDI note-off messages (as a series of numbers) for all note-ons which have not been matched by note-offs at its input.

Examples



When **midiflush** *receives a* bang, *it supplies note-offs for any held note-ons*

See Also

flush	Provide note-offs for held notes
midiin	Output received raw MIDI data
midiinfo	Set pop-up menu with names of MIDI devices
midiout	Transmit raw MIDI data
seq	Sequencer for recording and playing MIDI

midiformat

Prepare data in the form of a MIDI message

Input

Numbers received in the inlets are used as data for MIDI messages. The data is formatted into a complete MIDI message (with the status byte determined by the inlet) and sent out the outlet as individual bytes.

list In leftmost inlet: The first number is a pitch value and the second number is a velocity value, to be formatted into a note-on message.

In 2nd inlet: The first number is an aftertouch (pressure) value and the second number is a pitch value (key number), to be formatted into a polyphonic key pressure message.

In 3rd inlet: The first number is a control value and the second number is a controller number, to be formatted into a control message.

int In 4th inlet: The value is formatted into a program change message.

In 5th inlet: The value is formatted into an aftertouch (channel pressure) message.

In 6th inlet: The value is formatted into a pitch bend message.

In rightmost inlet: The number is stored as the channel number of the MIDI messages. The actual value of the status byte is dependent on the channel. Numbers greater than 16 are *wrapped around* to stay between 1 and 16.

float Converted to int.

Arguments

int Optional. Sets an initial value for the channel number of the MIDI messages. Numbers greater than 16 are *wrapped around* to stay between 1 and 16. If there is no argument, the channel number is initially set to 1.

float Converted to int.

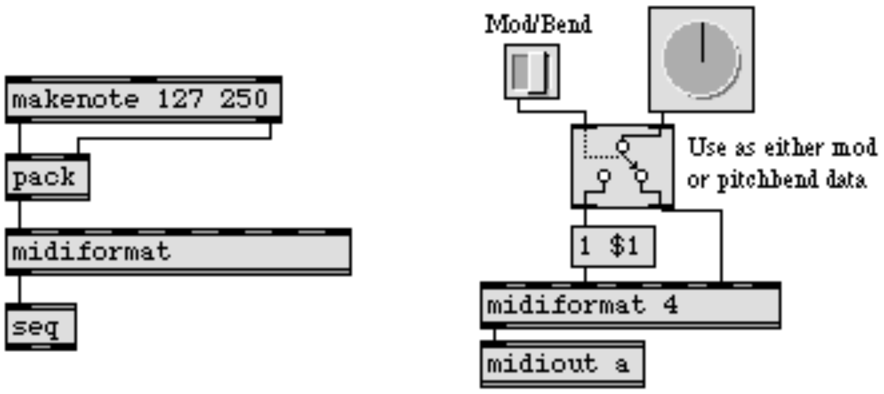
Output

int MIDI messages are sent out as individual bytes, for recording by the **seq** object or for transmission by the **midiout** object.

midiformat

Prepare data in the form of a MIDI message

Examples



Numbers are formatted into MIDI messages and sent out as individual bytes

See Also

borax	Report current information about note-ons and note-offs
midiinfo	Set pop-up menu with names of MIDI devices
midiout	Transmit raw MIDI data
midiparse	Interpret raw MIDI data
MIDI	MIDI overview and specification
Tutorial 34	Managing raw MIDI data

midiin

Output received raw MIDI data

Input

(MIDI)	**midiin** receives all MIDI messages from a MIDI input device.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming MIDI messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **midiin** object shows a pop-up menu for choosing a MIDI port or device.

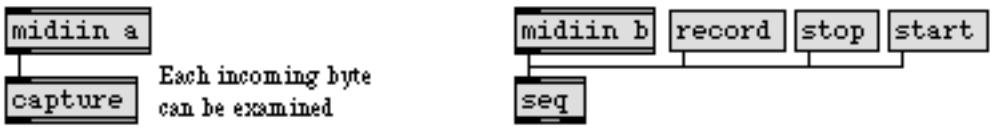
Arguments

a-z	Optional. Specifies the port from which to receive incoming MIDI messages. If there is no argument, **midiin** receives from port a (or the first input port listed in the MIDI Setup dialog.)

Output

int	All MIDI messages received from the specified port are sent out the outlet, byte-by-byte. Note that **midiin** does not "clean up" any use of running status in the incoming MIDI stream.

Examples



MIDI messages received in a port are output by a **midiin** *object*

See Also

midiout	Transmit raw MIDI data
midiparse	Interpret raw MIDI data
midiinfo	Set pop-up menu with names of MIDI devices
notein	Output received MIDI note messages
rtin	Output received MIDI real time messages
sysexin	Output received MIDI system exclusive messages
xnotein	Interpret MIDI note messages with release velocity
xbendin	Interpret extra precision MIDI pitch bend messages
Tutorial 34	Managing raw MIDI data
Using MIDI	Using Max with MIDI
MIDI	MIDI overview and specification
Ports	How MIDI ports are specified

midiinfo

Set pop-up menu with names of MIDI devices

Input

int — In left inlet: Causes **midiinfo** to send out a series of messages containing the names of the current MIDI *output* devices. Those messages can be used to set the individual items of a pop-up **umenu** object connected to the **midiinfo** object's outlet. The number received in the **midiinfo** object's left inlet is then sent in a set message to set the currently displayed **menu** item.

In right inlet: Causes **midiinfo** to send out a series of messages containing the names of the current *MIDI input* devices. Those messages can be used to set the individual items of a pop-up **umenu** object connected to the **midiinfo** object's outlet. The number received in the **midiinfo** object's left inlet is then sent in a set message to set the currently displayed **umenu** item, unless the number is less than zero, in which case no set message is sent.

bang — In left inlet: Same as int, but doesn't send a set message after setting the **umenu** items. The equivalent message to bang for retrieving input device names is -1 in the right inlet.

controllers — In left inlet: Causes **midiinfo** to send out a series of messages containing the names of all MIDI controllers (devices that transmit MIDI) in the current MIDI setup. Those messages can be used to set the individual items of a pop-up **umenu** object connected to the **midiinfo** object's outlet. The word controllers may be followed by a number, which sets the pop-up **umenu** to that item number after the menu items have been created.

Arguments

None.

Output

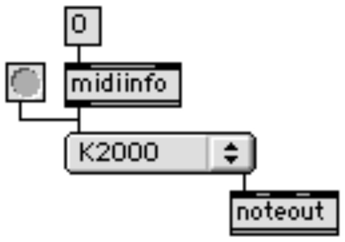
clear — **midiinfo** first sends a clear message out its outlet to clear all the receiving **umenu** object's items.

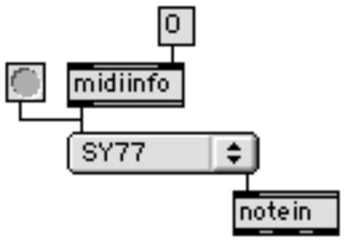
append — Immediately after sending the clear message, **midiinfo** sends an append message for each MIDI input or output device name, to set the items of a connected **umenu** object. The device names will be sent out in the order in which they appear in Max's MIDI Setup dialog.

set — If the incoming message to **midiinfo** is an integer greater than or equal to zero, a set message is sent after the append messages, to set the currently displayed menu item.

midiinfo

Examples





Get output device names for MIDI output objects *...and for MIDI input objects*

See Also

midiin	Output received raw MIDI data
midiout	Transmit raw MIDI data
umenu	Pop-up menu to display and send commands
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified

midiout

Transmit raw MIDI data

Input

int	The number is transmitted as a byte of a MIDI message to the specified port.
float	Converted to int.
list	The numbers are transmitted sequentially as individual bytes of a MIDI message to the specified port.
enable	The message enable 0 disables the object, causing it not to transmit MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI output port or device, specifies the port used to transmit the MIDI messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **midiout** object shows a pop-up menu for choosing a MIDI port or device.

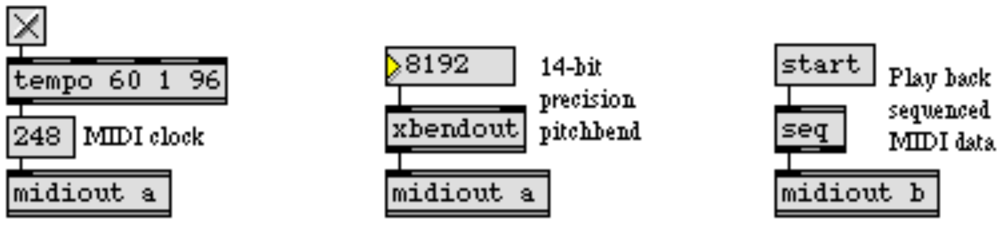
Arguments

a-z	Optional. Specifies the port for transmitting MIDI data. If there is no argument, **midiout** transmits out port a (or the first output port listed in the MIDI Setup dialog.)
(MIDI name)	Optional. The name of a MIDI output device may be used as the first argument to specify the port.

Output

(MIDI)	There are no outlets. The output is a byte of a MIDI message transmitted directly to the object's MIDI output port.

Examples



MIDI bytes received in the inlet are transmitted out the specified port

See Also

midiformat	Prepare data in the form of a MIDI message
midiin	Output received raw MIDI data
midiinfo	Set pop-up menu with names of MIDI devices
noteout	Transmit MIDI note messages
sxformat	Prepare MIDI system exclusive messages
xbendout	Format extra precision MIDI pitch bend messages
xnoteout	Format MIDI note messages with release velocity
Tutorial 34	Managing raw MIDI data
Using MIDI	Using Max with MIDI
MIDI	MIDI overview and specification
Ports	How MIDI ports are specified

midiparse

Interpret raw MIDI data

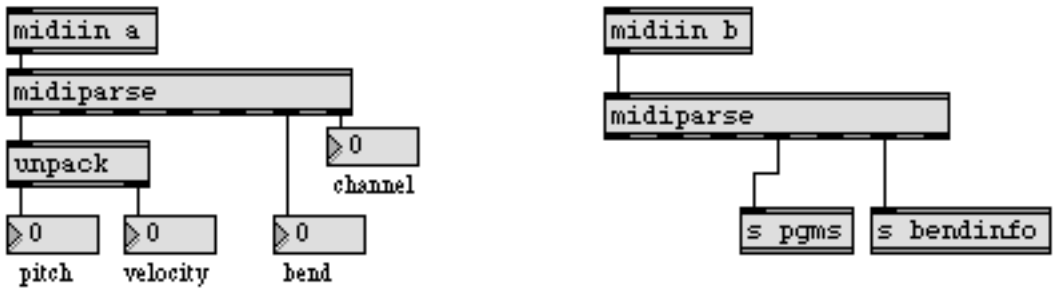
Input

int	Numbers received in the inlet are treated as bytes of a MIDI message (usually from a **seq** or **midiin** object). The status byte determines the outlet which will be used to output the data bytes.
float	Converted to int.
bang	Clears the **midiparse** object's memory of any partial MIDI message received up to that point.

Output

list	Out leftmost outlet: A note-on message. The first number is a pitch value and the second number is a velocity value.
	Out 2nd outlet: A polyphonic key pressure message. The first number is an after-touch (pressure) value and the second number is a pitch value (key number).
	Out 3rd outlet: A control message. The first number is a control value and the second number is a controller number.
int	Out 4th outlet: The number is a program change.
	Out 5th outlet: The number is an aftertouch (channel pressure) value.
	Out 6th outlet: The number is a pitch bend value.
	Out rightmost outlet: The number is the MIDI channel number.

Examples



Interpret the meaning of MIDI messages and filter different types of data

See Also

borax	Report current information about note-ons and note-offs
midiformat	Prepare data in the form of a MIDI message
midiin	Output received raw MIDI data
midiinfo	Set pop-up menu with names of MIDI devices
Tutorial 34	Managing raw MIDI data
MIDI	MIDI overview and specification

minimum

Output the smallest in a list of numbers

Input

int	In left inlet: If the number is less than the value currently stored in **minimum**, it is sent out the outlet. Otherwise, the stored value is sent out.
	In right inlet: The number is stored for comparison with subsequent numbers received in the left inlet.
float	Converted to int, unless there is a float argument, in which case all numbers are compared as floats.
list	In left inlet: The numbers in the list are all compared to each other, and the smallest value is sent out the outlet. The value stored in **minimum** is replaced by the *next smallest* value in the list. The **minimum** object accepts lists of up to 256 elements.
bang	In left inlet: Sends the most recent output out the outlet again.

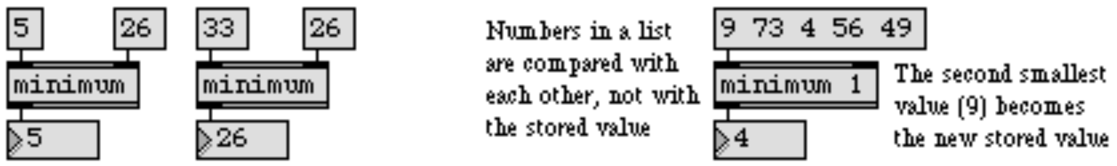
Arguments

int or float	Optional. Sets an initial value to be compared with numbers received in the left inlet. If the argument contains a decimal point, all numbers are compared as floats, and the output is a float. If there is no argument, the initial value is 0.

Output

int	The number received in the left inlet is compared with the value currently held by **minimum** (or numbers received as a list are compared with each other), and the smallest of the numbers is sent out the outlet.
float	Only if there is an argument with a decimal point.

Examples


5
26
minimum
5
33
26
minimum
26
Numbers in a list are compared with each other, not with the stored value
9 73 4 56 49
minimum 1
4
The second smallest value (9) becomes the new stored value

The output is the lesser of two numbers, or the smallest in a list of numbers

See Also

maximum	Output the greatest in a list of numbers
trough	If a number is less than previous numbers, output it
<	*Is less than*, comparison of two numbers

modifiers

Report modifier key presses on the computer keyboard

Input

(keyboard)	The keyboard input to **modifiers** comes directly from the computer keyboard.
bang	Sends out a report of the current modifier key states.
interval	The word interval followed by a number, specifies the rate, in milliseconds, used when polling the state of the modifier keys. A value of zero disables polling.

Arguments

int	Optional. Specifies a polling rate in milliseconds. The default value is 0 (no polling).

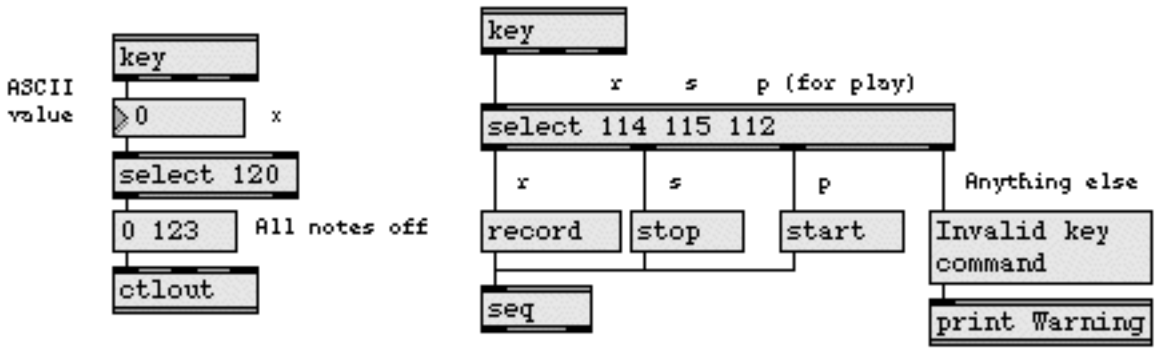
Output

int	Output is sent whenever a modifier key is pressed down on the computer keyboard. Modifier key states are reported as 0 (not pressed) or 1 (pressed). Out left outlet: The on/off state of the Shift key. Out second outlet: The on/off state of the Caps Lock key. Out third outlet: the on/off state of the Option key on Macintosh or the Alt key on Windows. Out fourth outlet: the on/off state of the Control key. Out fifth outlet: the on/off state of the Command key on Macintosh or the Control key on Windows. Note: The fourth and fifth outlets both report the on/off state of the Control key on Windows, since the Command key on Macintosh is equivalent to the Control key on Windows. For cross-platform uses, Windows users should use the fifth outlet of the **modifiers** object for reporting the Control key state. The fourth outlet also reports the Control key on Windows so that (older) Macintosh patches that use this key can be opened on Windows systems. The Macintosh Control key normally corresponds to the right-hand mouse button on Windows. See the section on file and key mappings in the Max Tutorials for a complete discussion of cross-platform keyboard issues.

modifiers

Report modifier key presses on the computer keyboard

Examples



Modifier keys typed on the computer keyboard can be used to trigger messages

See Also

key	Report key presses on the computer keyboard
keyup	Report key releases on the computer keyboard
numkey	Interpret numbers typed on the computer keyboard

mousefilter

Pass numbers only when the mouse button is up

Input

int	If the mouse button is up, the number is sent out the outlet. Otherwise, the number is ignored.

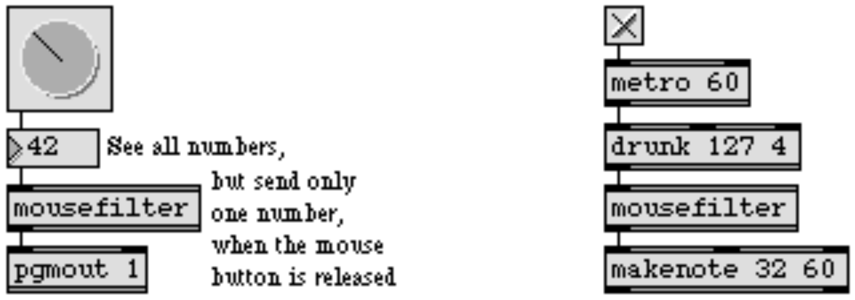
Arguments

None.

Output

int	The number received in the inlet is sent out only if the mouse button is up.

Examples


42
See all numbers, but send only one number, when the mouse button is released
mousefilter
pgmout 1
metro 60
drunk 127 4
mousefilter
makenote 32 60

Nothing gets through unless the mouse is up

See Also

mousestate	Report the status and location of the mouse
Tutorial 39	Mouse control

mousestate

Report the status and location of the mouse

Input

bang	Sends out the current horizontal and vertical coordinates of the location of the mouse, as well as the change in location since the last output.
poll	Causes **mousestate** to send out the mouse location, and the change in mouse location, *whenever the mouse is moved*, as well as when a bang is received. If poll is followed by the name of a graphics window, the coordinates returned by **mousestate** will be local to the graphics window, and only sent while the graphics window is visible.
nopoll	Undoes a poll message, reverting **mousestate** to its normal condition of waiting for a bang before reporting.
zero	Resets the point **mousestate** considers as the 0,0 point from which to measure the mouse location. The current location of the mouse is considered the new 0,0 point.
reset	Resets the 0,0 point to its default setting, in the upper left corner of the screen.

Arguments

None.

Output

int **mousestate** must have received at least one bang or poll message in its inlet before any output is sent out.

Out left outlet: Each time the mouse button is pressed, 1 is sent out. Each time the mouse button is released, 0 is sent out.

Out 2nd outlet: The horizontal location of the mouse, measured in terms of the number of pixels the mouse is to the right of the 0 point.

Out 3rd outlet: The vertical location of the mouse, measured in terms of the number of pixels the mouse is below the 0 point.

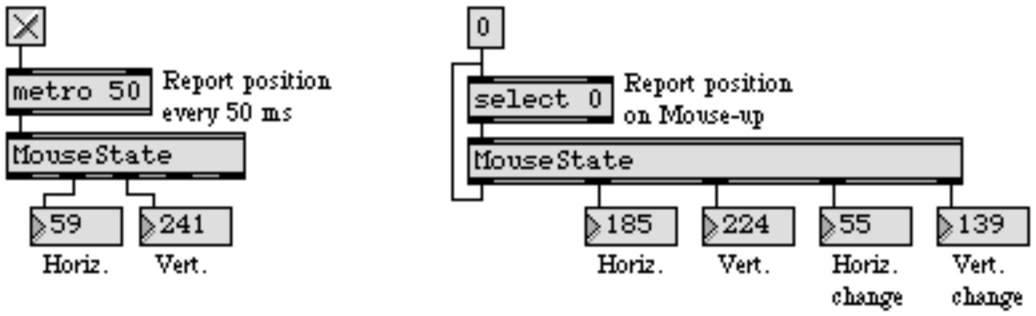
Out 4th outlet: The change in horizontal location of the mouse, since the last time the mouse location was reported.

Out right outlet: The change in vertical location of the mouse, since the last time the mouse location was reported.

mousestate

Report the status and location of the mouse

Examples



The mouse can provide continuous or discrete values

See Also

mousefilter	Pass numbers only when the mouse button is up
Tutorial 39	Mouse control

movie

Play a QuickTime movie in a window

Note: The **movie** object requires that QuickTime be installed on your system. If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.The **movie** object plays a QuickTime movie in its own window, and the **imovie** object plays a QuickTime movie in a box inside a patcher window.

Input

All messages below, recognized by the **movie** object, are similarly recognized by **imovie**.

int	Sets the current time location of the movie. If the movie is playing, it will play from the newly set location. 0 is always the beginning. The end time varies from one movie to another. (The length message reports the end time location out the left outlet.)
active	The word active, followed by a nonzero number, makes the movie active (the default). Followed by a 0, active makes the movie inactive. An inactive movie will not play or change location.
autofit	The word autofit, followed by a nonzero number, scales the movie to fit in the window currently displayed.
bang	Same as resume.
border	The word border, followed by a 0 or 1, toggles the movie's border type. The message border 1 (the default) uses the traditional Macintosh-style border for the movie window. The message border 0 displays only the rectangle in which the movie plays.
clear	Has the same effect as dispose with no arguments.
dispose	Closes the movie window if it is open, and removes all movies from the **movie** object's memory. If the word dispose is followed by the name of a loaded movie, only the named movie will be removed.
getrate	Reports the current rate multiplied by 65536 out the right outlet. Thus, normal speed is reported as 65536, half speed is reported as 32768, double speed backward is reported as -131072, etc. If the movie is not playing, the rate is reported as 0, and if no movie has yet been loaded nothing is sent out.
length	Reports the end time location of the movie.
loadintoram	The word loadintoram, followed by a nonzero number, attempts to load the entire movie into memory, if possible. The default is 0.
loop	The word loop, followed by a nonzero number, turns looping for the current film on. loop 0 (the default) disables looping.

loopend	The word loopend, followed by a number, sets the end point of a loop. The default value is corresponds to the end of the film.
loopset	The word loopset, followed by two numbers, sets the beginning and end points of a loop. the default values are 0 (i.e., the start of the film) for the start point and the end of the film for the endpoint.
loopstart	The word loopstart, followed by a number, sets the beginning point of a loop. The default value is 0 (i.e., the start of the film).

matrix The word matrix, followed by nine floating point numbers, reloads the current movie into RAM after performing a transformation matrix operation on the image. This transformation is the same one used for the mapping in QuickTime of points from one coordinate space (i.e, the original image) into another coordinate space (a scaled, rotated, or translated version of the original image).

The transform matrix operation consists of nine matrix elements

$$\begin{array}{ccc} a & b & u \\ c & d & v \\ t\_x & t\_y & w \end{array}$$

if *u* and *v* are 0., and *w* is 1., we have the following translation formula.

$x' = a*x + c*y + t\_x;$

$y' = b*x + d*y + t\_y$

The following formulas are used for scaling/rotation:

$a=xscale*cos(\theta)$

$b=yscale*sin(\theta)$

$c=xscale*(-sin(\theta))$

$d=yscale*cos(\theta)$

For more on the transformation matrix, consult the Apple QuickTime Developer documentation found at:

http://developer.apple.com/techpubs/quicktime/qtdevdocs/INMAC/QT/iqMovieToolbox.c.htm#18006

mute The word mute, followed by a nonzero number, turns off the movie's sound (if it has any). Followed by a 0, mute turns on the movie's sound (the default).

next	The word next, followed by a number, moves the time location ahead by that amount. If no number is supplied, next moves the time ahead by 5. (The actual time meaning of these units varies from movie to movie.)
nextmovie	Stops the movie if it is playing, and switches to the movie that was loaded just prior to the current movie. (The movies are stored in reverse order from the order in which they were loaded.) If there is no prior movie, nextmovie *wraps around* back to the most recently loaded movie. Note that the title of the movie window is not automatically changed, even though the "current movie" has been changed by nextmovie.
open	Brings the movie window to the foreground (applies only to **movie**, not **imovie**).
passive	The word passive, followed by a nonzero number, sets the passive mode. In passive mode, starting a movie will not cause the frame to change unless a bang message is received. passive 0 (the default) sets the movie object to respond to normal start messages.
pause	Stops the movie.
prev	The word prev, followed by a number, moves the time location backward by that amount. If no number is supplied, prev moves the time backward by 5.
quality	The word quality, followed by a number, sets the minimum interval, in milliseconds, between movie redraws. The default is 0 (i.e., no minimum).
rate	The word rate, followed by one or more integers or floats, sets the playing speed of the movie. If rate is followed by one integer, that number is taken to be a whole number playing speed. If rate is followed by two numbers, the first number is taken to be the numerator and the second the denominator of a fractional speed. 1 is the normal playing speed, 0 means the movie is stopped, and a negative rate plays backwards. rate 1 2 would play the movie at half speed. Immediately after you send a non-zero rate message, the movie will begin playing, so you may wish to precede any rate messages with an integer to locate to the desired starting position.
read	The word read, followed by a symbol, looks for a QuickTime movie file with that name in Max's file search path, and opens it if it exists, displaying the movie's first frame in a movie window. If the filename contains any spaces or special characters, the name should be enclosed in double quotes or each special character should be preceded by a backslash (\). The word read by itself puts up a standard Open Document dialog box and reads in any movie file you select. The read message will open at least 26 different types of files that can be opened by QuickTime, these include movie files such as MPEG, audio files including AIFF and MP3, and graphics files including GIF and JPEG.
readany	The readany message opens any type of file, using QuickTime routines to try to interpret it as a movie or other supported media file.

rect	The word rect, followed by four numbers, specifies the size of the rectangle in which the movie is displayed within the movie window. The first two numbers specify the position of the rectangle within the movie window, in relative coordinates, and the second two numbers specify the width and height, in pixels, of the rectangle.
resume	Begins playing the movie from its current location, at the most recently specified rate.
start	Sets the movie's rate to 1 and begins playing from the beginning. If the word start is followed by the name of a specific loaded movie, that movie becomes the current movie before starting.
startat	The word switch, followed by a number, set the current time location of the movie and begins playing from that point.
stop	Stops the movie.
switch	The word switch, followed by a symbol, make the named movie the active one without changing the transport state (See the start message).
time	Reports the current time location of the movie.
title	Sets the title of the movie window to the name of the current movie. This is necessary in conjunction with the nextmovie message (or a start message specifying a different movie) if you want the title of the movie window to show the name of the current movie correctly. You can set the title of the movie window to any text you want, using the message title followed by a symbol.
vol	The word vol, followed by a number, sets the movie's sound volume. Any number less than 1 mutes the sound. The maximum volume is 255.
wclose	Closes the movie window.
windowpos	The word windowpos, followed by four numbers, specifies the location and size of the movie window on the screen. The four numbers specify the left, top, right, and bottom of the movie window in global coordinates. This message is only supported by the **movie** object, not the **imovie** object.

Arguments

symbol	Optional. Specifies the name of a QuickTime movie file to be read into **movie** automatically when the patch is loaded. The same effect can be achieved for **imovie** by selecting the object in an unlocked patcher and choosing Get Info... from the Object menu to select a movie file. Both objects retain the name(s) of the movie(s) they have loaded at the time that the patch is saved, and attempt to load the same movie(s) the next time the patch is opened.

Output

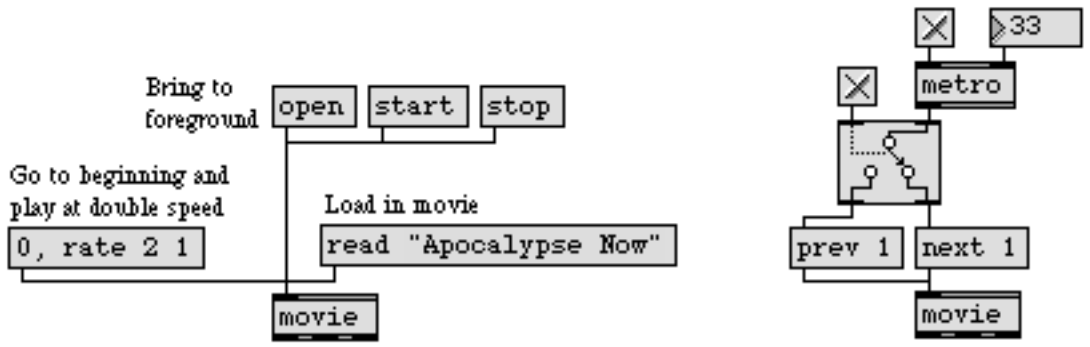
int Out left outlet: The current time location, when a time message is received; the end time location when a length message is received.

Out middle outlet: The horizontal position of the mouse, relative to the left side of the movie box or window, when the mouse is clicked or dragged inside the movie.

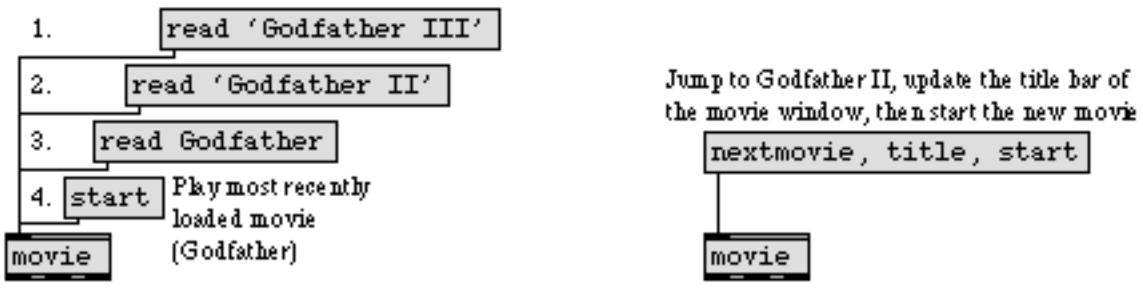
Out right outlet: The vertical position of the mouse, relative to the top of the movie box or window, when the mouse is clicked or dragged inside the movie.

Also, in response to a getrate message, the current movie rate multiplied by 65536 is sent out the right outlet.

Examples



Play a QuickTime movie, or move through it in a variety of ways


1.
read 'Godfather III'
2.
read 'Godfather II'
3.
read Godfather
4.
start
Play most recently loaded movie (Godfather)
movie
Jump to Godfather II, update the title bar of the movie window, then start the new movie
nextmovie, title, start
movie

Hold multiple movies (which are stored in reverse order from the order received)

See Also

imovie	Play a QuickTime movie in a patcher window

Input

record — In left inlet: Begins recording all messages received in the other inlets. The word record, followed by one or more track numbers, begins recording those tracks.

In other inlets: Begins recording messages on the track that corresponds to the inlet.

play — In left inlet: Plays back all messages recorded earlier, sending them out the corresponding outlets in the same rhythm and at the same speed they were recorded. The word play, followed by one or more track numbers, begins playing those tracks.

In other inlets: Plays back all messages on the track that corresponds to the inlet.

stop — In left inlet: Stops **mtr** when it is recording or playing. The word stop, followed by one or more track numbers, stops those tracks.

In other inlets: Stops the track that corresponds to the inlet.

next — In left inlet: Causes each track to output only the next message in its recorded sequence. When a next message is received, the track number and the *delta time* of each message being output are sent out the leftmost outlet as a list. The word next, followed by one or more track numbers, outputs the next message stored in those tracks.

In other inlets: Outputs the next message stored on the track that corresponds to the inlet.

rewind — In left inlet: Resets **mtr** to the beginning of its recorded sequence. This command is used to return to the beginning of the sequence when stepping through messages with next. To return to the beginning of a sequence while playing or recording, just repeat the play or record message. When **mtr** is playing or recording, a stop message should precede a rewind message. The word rewind, followed by one or more track numbers, returns to the beginning of those tracks.

In other inlets: Returns the pointer to the beginning of the track that corresponds to the inlet.

mute — In left inlet: Causes **mtr** to stop producing output, while still continuing to "play" (still moving forward in the sequence). The word mute, followed by one or more tracks, mutes those tracks.

In other inlets: Mutes the track that corresponds to the inlet.

delay — In left inlet: The word delay, followed by a number of milliseconds, sets the first *delta time* value of each track to that number, so that all tracks begin playing back that amount of time after the play message is received.

In other inlets: Sets the initial *delta time* of the track that corresponds to the inlet.

first — In left inlet: The word first, followed by a number of milliseconds, causes **mtr** to wait that amount of time after a play message is received before playing back. Unlike delay, first does not alter the delta time value of the first event in a track, it just waits a certain time (in addition to the first delta time) before playing back from the beginning.

write — In left inlet: Calls up the standard Save As dialog box, allowing the contents of **mtr** to be saved as a separate file. Note that the only way to save the contents of **mtr** is with the write message; the object's contents cannot be embedded in a patcher file.

In other inlets: Writes a file containing only the track that corresponds to the inlet.

read — In left inlet: Calls up the standard Open Document dialog box, so that a previously saved file can be read into **mtr**.

In other inlets: Opens a file containing only the track that corresponds to the inlet.

int — In any inlet other than the left inlet: If the track is currently being recorded, numbers received in that track's inlet are combined with a *delta time* (the number of milliseconds elapsed since the previous event) and stored in **mtr**.

list — In any inlet other than the left inlet: If the track is currently being recorded, lists received in that track's inlet are stored in **mtr**, preceded by the delta time.

any symbol — In any inlet other than the left inlet: If the track is currently being recorded, symbols received in that track's inlet are stored in **mtr**, preceded by the delta time.

Although **mtr** can record individual bytes of MIDI messages received from **midiin**, it stores each byte with a separate delta time, and does not format the MIDI messages the way **seq** does. If you want to record complete MIDI messages and edit them later, **seq** is better suited for the task. On the other hand, **mtr** is perfectly suited for recording sequences of numbers, lists, or symbols from virtually any object in Max: specialized MIDI objects such as **notein** or **pgmin**, user interface objects such as **number box**, **slider**, and **dial**, or any other object.

In order for a file to be read into **mtr** for playback, it must be in the proper format. An **mtr** multi-track sequence can even be typed in a text file, provided it adheres to the format. The contents of the different tracks are listed in order in an **mtr** file, and the format of each track is as follows. Note that a semicolon (;) ends each line.

Line 1:	track <track number>;	(Track in which to store subsequent data)
Line 2, etc.:	<delta time> <message>;	
Last line:	end;	(End of this track's data)

clear In left inlet: Erases the contents of **mtr**. The word clear, followed by one or more track numbers, clears those tracks.

In other inlets: Erases the track that corresponds to the inlet.

unmute In left inlet: Undoes any previously received mute messages. The word unmute, followed by one or more track numbers, unmutes those tracks.

In other inlets: Unmutes the track that corresponds to the inlet.

Arguments

int Optional. Specifies the number of tracks in the **mtr**. The number of tracks determines the number of inlets and outlets *in addition to* the leftmost inlet and outlet. Up to 32 tracks are possible. If there is no argument, there will be only one track.

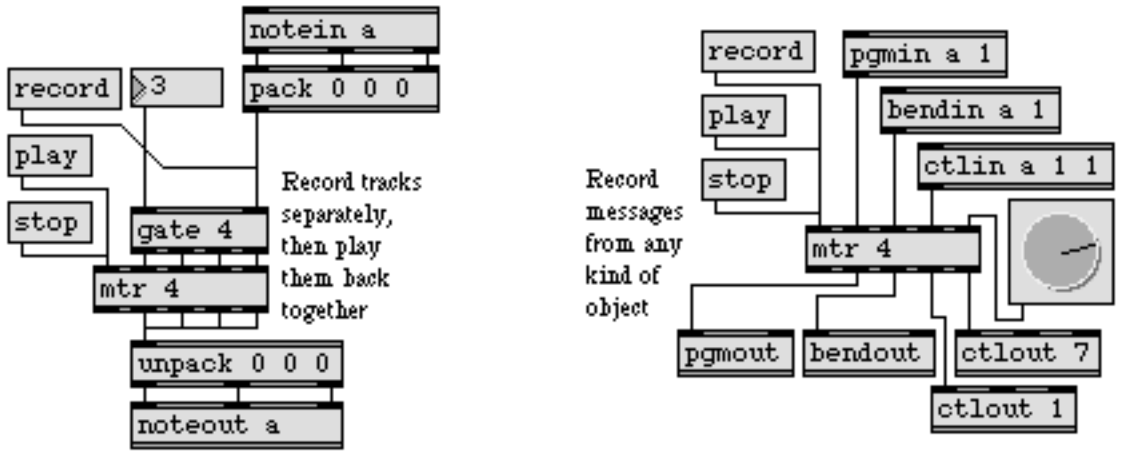
Output

anything Out all track outlets: When a play message is received in the leftmost inlet, the messages stored in each track are sent out the outlet of that track, in the same rhythm and at the same speed they were recorded. A play message received in the inlet of an individual track plays that particular track.

When a next message is received in the leftmost inlet, the next message in each track is sent out its corresponding outlet. The word next, received in the inlet of an individual track, sends out the next message in that track.

list Out left outlet: Whenever a value is sent out in response to a next message, the track number and delta time of that value are sent out the left outlet as a two-item list.

Examples



Record MIDI data or other events

See Also

hslider	Output numbers by moving a slider onscreen
multislider	Multiple slider and scrolling display
seq	Sequencer for recording and playing MIDI
timeline	Time-based score of Max messages
rslider	Display or change a range of numbers
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials
Tutorial 36	Multi-track sequencing
Sequencing	Recording and playing back MIDI performances

multislider

Multiple slider and scrolling display

Input

int — Sets all slider values and positions to the number received and outputs a list reflecting the current values. If the **multislider** data type is set to float, the values in the incoming list are converted to floats.

float — Sets all slider values and positions to the number received and outputs a list reflecting the current values. If the **multislider** data type is set to int, the values in the incoming list are truncated and converted to ints.

list — Sets each slider to a corresponding value in the list from left to right, with the first value in the list setting the first slider. If the **multislider** has a different number of sliders than is present in the list, the number of sliders is changed to the number of items in the list. In such a case, the outside dimensions of the **multislider** will not change, only the width or height of the sliders.

bang — Outputs the current slider values as a list.

border — The word border, followed by an integer, tells a **multislider** which of its outside borders to draw. This is useful for placing **multislider** objects next to each other.

It is both easier and more customary to use the Inspector to set the colors for the border. The arguments to border are:

border 0	Draw no borders
border 1	Draw left border
border 2	Draw right border
border 4	Draw top border
border 8	Draw bottom border

Any combination of borders can be drawn by adding these values. For example, border 15 draws all borders.

brgb — The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **multislider** object. The default value is white (brgb 255 255 255).

contdata — The word contdata, followed by a one or zero, allows continuous output mode to be turned on and off for non-scrolling display styles. If this mode is turned on, the **multislider** object will output a list of its current slider values each time the mouse is clicked and dragged. If this mode is turned off, the **multislider** will only output a list when the mouse button is pressed and when it is released. The continuous output mode can also be set using the Inspector.

displayonly — Toggles display only mode on and off. When display only mode is on, the **multislider** object will not allow user interaction with the display. The default is off (0).

echo	Toggles echo mode on and off. When echo mode is on, the **multislider** object will output any list received in its inlet. The default is off (0).
fetch	The word fetch, followed by a number, sends the value of the numbered slider out the rightmost outlet.
frgb	The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the slider color of the **multislider** object. The default value is black (frgb 0 0 0).
interp	The word interp, followed by a one or zero, enables or disables interpolation mode. When interpolation mode is on (the default), the **multislider** object will output interpolated values when a slider is moved. In most cases you probably will not want to disable interpolation mode.
max	Sets all sliders to their maximum values.
maximum	The word maximum, followed by an integer or floating point value, sets the maximum range value for the multislider object. The default is 1.0 when using floating point sliders, and 127 when using integer sliders. This value can also be set using the Inspector.
min	Sets all sliders to their minimum values.
minimum	The word minimum, followed by an integer or floating point value, sets the minimum range value for the **multislider** object. The default is -1.0 when using floating point sliders, and 0 when using integer sliders. This value can also be set using the Inspector.
(mouse)	The way that a **multislider** responds to the mouse is determined by its chosen display style (see Arguments, below). A **multislider** will respond to mouse clicks when its display style is *non-scrolling* (Thin Line or Bar). Clicking on a forward or reverse scrolling display **multislider** (Point Scroll or Line Scroll) has no effect. If continuous output mode is enabled, the list of the current values will be sent out each time the mouse moves while dragging. If the continuous output mode is off, this list is only sent out when the mouse button is pressed or released. The continuous output option can be set in the **multislider** object's Inspector. When the display style is non-scrolling, clicking on any slider in a **multislider** immediately positions the slider at the click point. The current value of all sliders is sent out. Dragging across a **multislider** will set the other sliders in the same manner. If continuous output mode is enabled, the list of the current values will be sent out each time the mouse moves while dragging. If the continuous output mode is off, this list is only sent out when the mouse button is pressed or released. The continuous output option can be set in the **multislider** object's Inspector. If the mouse is moved quickly across a range of sliders, the mouse's position is likely not to be polled quickly enough by the computer to provide a value for each

	and every slider it appears to pass. By default, **multislider** will automatically interpolate slider values between successively polled mouse positions. You can use the interp message to disable interpolation, if desired.
peakhold	The word peakhold, followed by a one or zero, enables or disables peak hold mode. When peak hold mode is on, the peak value of each slider is represented by a thin line, whose color can be set in the **multislider** object's Inspector. the peak values may be reset with the peakreset message.
peakreset	Resets the current peak values to the current slider values.
quantiles	In left inlet: The word quantiles, followed by a list of floats between 0 and 1.0, multiplies each list element by the sum of all the values in the **multislider**. This result is then divided by 2^{15} (32,768). Then, **multislider** sends out the address at which the sum of all values up to that address is greater than or equal to the result for each list element.
rgb2	The word rgb2, followed by three numbers between 0 and 255, sets the RGB values for the peak indicators when Peak-Hold display is turned on (see peakhold and peakreset messages). The default value is grey (rgb2 127 127 127). The color can also be set using the Inspector.
select	Selectively sets slider values. For example, select 1 30 2 4 5 50 sets the first slider to 30, the second to 4, and the fifth slider to 50 (the top or leftmost slider is always number 1).
set	The word set, followed by a slider number and a value, sets the numbered slider to that value without triggering any output.
setborder	The word setborder, followed by four integers representing the left, right, top and bottom borders of the **multislider** object, set the object's borders. It is similar in function to the border message (see above). A 0 indicates that the specified border segment will not be drawn, and a 1 draws the border. The default is to draw all borders (setborder 1 1 1 1).
setminmax	The word setminmax, followed by two floats or two integers, sets the low and high range values for the **multislider** object. The default values are -1.0 and 1.0 for floating point sliders and 0 and 127 for integer sliders.

setstyle The word setstyle, followed by an int in the range 0-5, sets the display style of the **multislider** object. The default value is Thin Line (setstyle 0). The display style values are:

setstyle 0	Thin line
setstyle 1	Bar
setstyle 2	Point Scroll
setstyle 3	Line Scroll
setstyle 4	Reverse Point Scroll
setstyle 5	Reverse Line Scroll

When the display style is set to Thin Line or Bar, each slider displays its current value as a thin line. When one of the other (scrolling) display styles is chosen, each slider provides a continuously scrolling display of its current and most recent past values. (The number of past values shown is determined by the display size of the **multislider**, in pixels.)

Note: A scrolling display **multislider** may not be able to update at the rate it receives data. This can result in some data points not being displayed.

settype The word settype, followed by a 0 or 1, sets the **multislider** object for integer (0) or floating point (1) operation. The Inspector can also be used to set the **multislider** object's type. The default is integer (settype 1).

size The word size, followed by a number, sets the number of sliders the **multislider** object has. The default is 1, and the maximum number of sliders is 4096.

sum Outputs a sum of all current slider values as a float.

Inspector

The behavior of a **multislider** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **multislider** object displays the **multislider** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **multislider** Inspector lets you set the following attributes:

- *Slider Range Minimum* and *Maximum values*. The default Min. value is -1. The default Max. value is 1.

- *Number of Sliders*. The maximum number of sliders a **multislider** object can have is 4096, and the default is 1. You can also choose Integer or Floating Point sliders. The default is floating point.

- *Slider Style*. You can choose Thin line, Bar, Point Scroll, Line Scroll, Reverse Point Scroll, or Reverse Line Scroll styles. When the display style is set to Thin

Line (the default) or Bar, each slider displays its current value as a thin line. When one of the other (scrolling) display styles is chosen, each slider provides a continuously scrolling display of its current and most recent past values. (The number of past values shown is determined by the display size of the **multislider**, in pixels.) You can also select Continuous Data Output and Peak Hold display modes (the default is off for both modes).

- *Orientation* lets you choose horizontal or vertical (default) data display.

- The *Draw Borders* checkboxes let you specify borders for all four sides of the **multislider** object.

- The *Color* option lets you use a swatch color picker or RGB values to specify colors for the Sliders, Background and Peak Indicators of the **multislider** object. The default color for the sliders is 0 0 0, the default background color is 255 255 255, and the default peak indicator color is 127 127 127.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

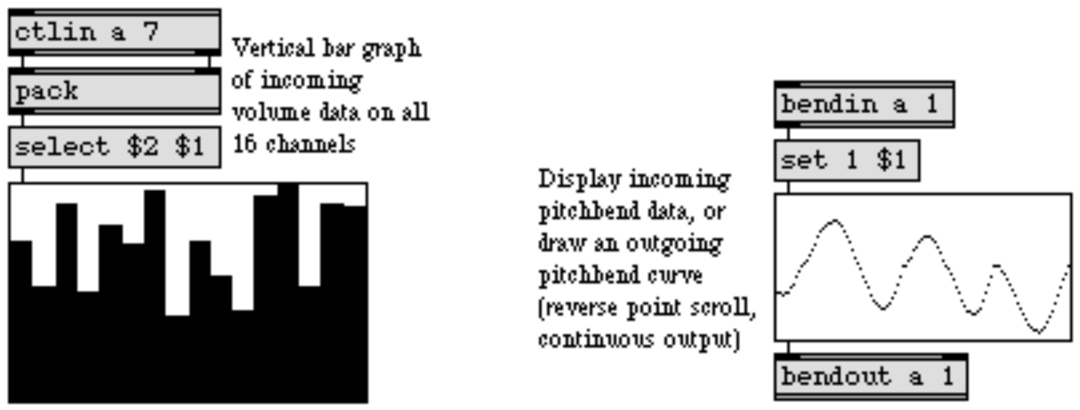
Arguments

None.

Output

list	Out left outlet: When a **multislider** receives a list, int, or float in its inlet, it outputs a list of its current values. The list is also sent out when the sliders are changed with the mouse.
int or float	Out right outlet: The value of a numbered slider specified by the fetch message. The output reflects the current data type settings (see the settype message).

Examples


ctlin a 7
pack
select $2 $1
Vertical bar graph of incoming volume data on all 16 channels
bendin a 1
set 1 $1
Display incoming pitchbend data, or draw an outgoing pitchbend curve (reverse point scroll, continuous output)
bendout a 1

multislider *drawing styles*

See Also

dial	Output numbers by moving a dial onscreen
hslider	Output numbers by moving a slider onscreen
kslider	Output numbers from a keyboard onscreen
matrixctrl	Matrix-style switch control
pictctrl	Picture-based control
pictslider	Picture-based slider
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

next

Input

anything	Messages to be tested to determine whether they are part of the same logical event. A logical event is one of the following: a mouse click, the ongoing polling of a mouse drag, an event generated by the scheduler (such as the bang from a **metro**), a MIDI event, or a keyboard event. **next** determines whether the current message is part of the same event as the previously received message. For example, if you click on a bang twice, the two bangs are not part of the same logical event. But if you put bang, bang in a message box, or use the **uzi** object to send out two bangs in a row, these bangs are part of the same logical event.

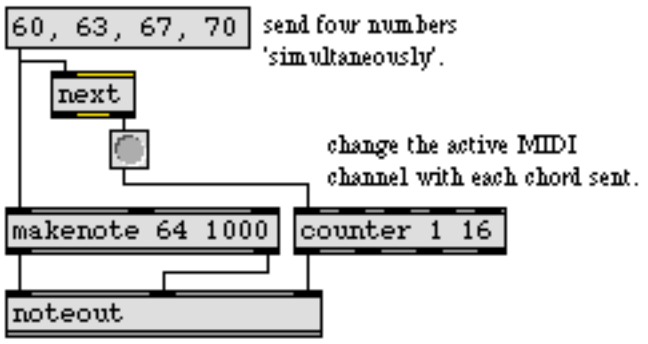
Arguments

None.

Output

bang	Out left outlet: A bang is sent out if the current message is not part of the same logical event as the previously received message.
	Out right outlet: A bang is sent out if the current message is part of the same logical event as the previously received message.

Examples


60, 63, 67, 70
send four numbers 'simultaneously'.
next
change the active MIDI channel with each chord sent.
makenote 64 1000
counter 1 16
noteout

next *detects when separate Max messages occur within the same logical event.*

See Also

uzi	Send a specific number of bang messages
defer	De-prioritize a message
delay	Delay a bang before passing it on
Messages	Using the comma in a message box]

notein

Input

(MIDI)	**notein** receives its input from a MIDI note-on or note-off message received from a MIDI input device.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming note messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **notein** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

a-z	Optional. Specifies the port from which to receive incoming note messages. If there is no argument, **notein** receives from all channels on all ports.
(MIDI name)	Optional. The name of a MIDI input device may be used as the first argument to specify the port.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to receive note messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

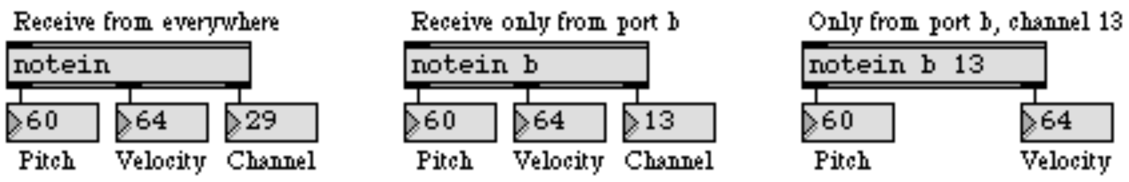
Output

int	Out left outlet: The number is the pitch value of the incoming note message.
	Out 2nd outlet: The number is the velocity of the incoming note-on message if non-zero, 0 for a note-off message. To receive release velocity, use **xnotein**.
	If a specific channel number is included in the argument, there are only two outlets. If there is no channel number specified by the argument, **notein** will have a third outlet, on the right, which will output the channel number of the incoming note message.

notein

Output received MIDI note messages

Examples



Note-on messages can be received from everywhere, a specific port, or a specific port and channel

See Also

ctlin	Output received MIDI control values
midiin	Output received raw MIDI data
noteout	Transmit MIDI note messages
rtin	Output received MIDI real time messages
xbendin	Interpret extra precision MIDI pitch bend messages
xnotein	Interpret MIDI note messages with release velocity
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified
Tutorial 12	Sending and receiving MIDI notes

noteout

Transmit MIDI note messages

Input

int	In left inlet: The number is the pitch value of a MIDI note message transmitted on the specified channel and port. Numbers are limited between 0 and 127.
	In middle inlet: The number is stored as the velocity of a note message, to be used with pitch values received in the left inlet. Numbers are limited between 0 and 127. 0 is considered a note-off message, 1-127 are note-on messages.
	In right inlet: The number is stored as the channel number on which to transmit the note-on messages.
float	Converted to int.
list	In left inlet: The first number is used as the pitch, the second number is used as the velocity, and the third number is used as the channel, of a transmitted MIDI note message.
enable	The message enable 0 disables the object, causing it not to transmit MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	In left inlet: The word port, followed by a letter a-z or the name of a MIDI output port or device, specifies the port used to transmit the MIDI messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **noteout** object shows a pop-up menu for choosing a MIDI port or device.

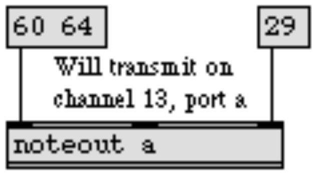
Arguments

a-z	Optional. Specifies the port for transmitting MIDI note messages. Channel numbers greater than 16 received in the right inlet will be *wrapped around* to stay within the 1-16 range. If there is no argument, **noteout** initially transmits out port a, on MIDI channel 1.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to transmit note messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
(MIDI name)	Optional. The name of a MIDI output device may be used as the first argument to specify the port.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

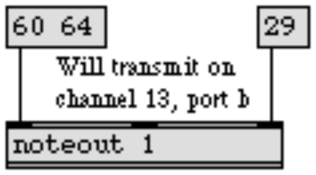
Output

(MIDI) There are no outlets. The output is a MIDI note-on message transmitted directly to the object's MIDI output port.

Examples



Letter argument transmits to only one port



Otherwise, number specifies both port and channel

See Also

ctlout	Transmit MIDI control messages
midiout	Transmit raw MIDI data
notein	Output received MIDI note messages
xbendout	Format extra precision MIDI pitch bend messages
xnoteout	Format MIDI note messages with release velocity
Ports	How ports are specified
Tutorial 12	Sending and receiving MIDI notes

number box

Display and output a number

Input

int or float — The number received in the inlet is stored and displayed in the **number box** and sent out the outlet. A float is converted to int by an int **number box**, and vice versa.

When the active patcher window is locked, numbers can be entered into a **number box** by clicking on it with the mouse and typing in a number on the computer keyboard. Typing the Return or Enter keys on Macintosh or the Enter key on Windows, or clicking *outside* the **number box**, sends the number out the outlet.

Dragging up and down on the **number box** with the mouse (when the patcher window is locked) moves the displayed value up and down, and outputs the new values continuously. In the float **number box**, dragging to the left of the decimal point changes the value in increments of 1. Dragging to the right of the decimal point changes the fractional part of the number in increments of 0.01.

bang — Sends the currently displayed number out the outlet.

brgb — The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **number box**. The default value is white (brgb 255 255 255).

color — The word color, followed by a number from 0 to 15, sets the background of the number box to one of the standard object colors which are also available via the Color submenu in the Object menu.

flags — The word flags, followed by a number, sets characteristics of the appearance and behavior of the **number box**. The characteristics (which are described on the next page, under *Arguments*) are set by adding together specific numbers to designate the desired characteristics, as follows: 4=Bold type, 16=Hexadecimal display, 32=No triangle, 64=Send on mouse-up only, 128=Can't change with mouse, 256=MIDI C3 display, 1024=Roland octal display, 2048=Binary display, 4096=MIDI C4 display, 8192 =Transparent display mode (useful for displaying and editing numbers over other objects). So, for example, flags 180 (4+16+32+128=180) will set the **number box** to display its numbers in hexadecimal format, in bold type, with no triangle, and unchangeable by the mouse.

frgb — The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the number values displayed by the **number box**. The default value is black (brgb 0 0 0).

max — The word max, followed by a number, sets the maximum value that can be displayed or sent out by the **number box**. The word max by itself sets the maximum to None (removes a prior maximum value constraint).

min — The word min, followed by a number, sets the minimum value that can be displayed or sent out by the **number box**. The word min by itself sets the minimum to None (removes a prior minimum value constraint).

rgb2 The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the number values displayed by the **number box** when it is highlighted or being updated. The default value is black (brgb 0 0 0).

rgb3 The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **number box** when it is highlighted or being updated. The default value is white (brgb 255 255 255).

set The word set, followed by a number, sets the stored and displayed value to that number without triggering output.

(typing) When a **number box** is *highlighted* (indicated by a filled-in triangle) in a patcher window, numerical keyboard input is sent to the **number box** to change its value. Clicking the mouse or pressing Return on Macintosh or Enter on Windows stores a pending typed number.

(Font menu) The font and size of a **number box** can be altered by selecting it and choosing a different font or size from the Font menu.

Inspector

The behavior of a **number box** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **number box** object displays the **number box** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **number box** Inspector lets you set the following attributes:

You can set the range for stored, displayed, typed, and passed-through values by typing values into the *Range Min.* and *Max.* boxes. If the *No Min.* and *No Max.* checkboxes are checked (the default state), the **number box** objects will have their minimum and maximum values set to "None." Unchecking these boxes sets the minimum and maximum values to 0.

The Options section of the Inspector lets you set the display attributes of the **number box**. Other options available in the **number box** Inspector window are: *Bold* (to display in bold typeface), *Draw Triangle* (to have an arrow pointing to the number, giving it a distinctive appearance), *Output Only on Mouse-Up* (to send a number only when the mouse button is released, rather than continuously), *Can't Change* (to disallow changes with the mouse or the computer keyboard), and *Transparent* (to display only the number in the **number box** and not the box, so that the number box resembles a **comment** object).

The *Display Style* pop-up menu lets you select the way that number values are represented. *Decimal* is the default method of displaying numbers. *Hex* shows numbers in hexadecimal, useful for MIDI-related applications. *Roland Octal* shows numbers in a format used by some hardware devices where each digit ranges from 1 to 8; 11 is 0 and 88 is 63. *Binary* shows numbers as ones and zeroes. *MIDI Note Names* shows numbers according to their MIDI pitch value, with 60 displayed as C3. *Note Names C4* is the same as *MIDI Note Names* except that 60 is displayed as C4. With all display modes, numbers must be typed in the format in which they are displayed.

The *Color* option lets you use a swatch color picker or RGB values used to display the number box and its background in its normal and highlighted forms. *Number* sets the color for the number displayed (default 0 0 0), *Background* sets the color for the number box object itself (default 221 221 221), *Highlighted Number* sets the color of the number display when the number box is selected or its values are being updated (default 222 222 222), and *Highlighted Background* sets the color of the number box when it is highlighted or being updated (default 0 0 0).

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

Output

int or float The number displayed in the **number box** is sent out the outlet. Numbers received in the inlet or typed on the computer keyboard can exceed the limits of the **number box**, but the value that gets stored, displayed, and sent out will automatically be limited to the specified range.

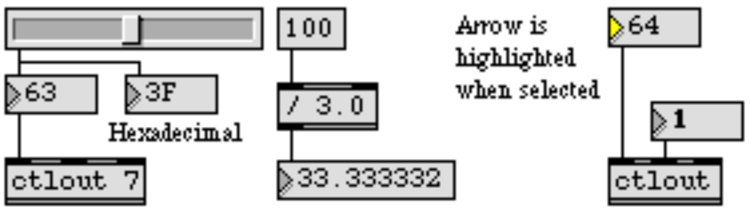
The **number box** does not resize itself automatically according to the size of the number it contains. If the number received is too long to be displayed in the **number box**, it is displayed in abbreviated form followed by an ellipsis (…) in the case of an int **number box**, or as a plus sign (+) in the case of a float **number box**.

The number is stored and sent out of the **number box** as usual, despite this abbreviated display.

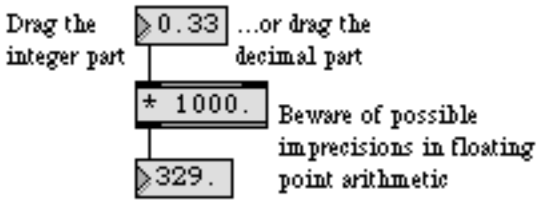
number box

Display and output a number

Examples



Displays numbers passing through



Drag the integer part
0.33
...or drag the decimal part
* 1000.
Beware of possible imprecisions in floating point arithmetic
329.

Can be used to output numbers

See Also

float	Store a decimal number
int	Store an integer value
Tutorial 3	About numbers
Tutorial 10	Number boxes

numkey

Interpret numbers typed on the computer keyboard

Input

int The number is an ASCII value received from a **key** or **keyup** object. When digits are typed on the computer keyboard, **numkey** recognizes the ASCII values and interprets them as the numbers being typed.

The keys recognized by **numkey** are the digits 0-9, the Delete (Backspace) key, decimal point (period), Return, and Enter. Digits are combined as a single number and stored in **numkey**.

bang Sends the number currently stored in **numkey** out the left outlet, and resets the stored number to 0.

clear Resets the stored number to 0.

Arguments

Optional. A float argument causes **numkey** to understand the decimal point and the fractional part of a number, and send out floats instead of ints. (The argument does not, however, set an initial value for **numkey**. The initial value is always 0.)

Output

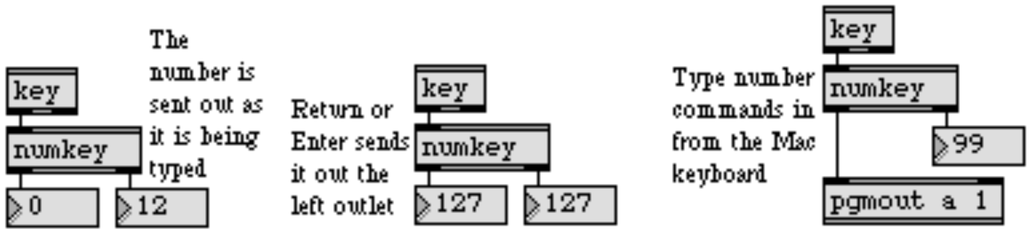
int When digits are typed on the computer keyboard, and the ASCII value (from **key** or **keyup**) is received in the inlet, the digits are combined as a single number and stored in **numkey**. The stored number is sent out the right outlet each time a new digit is typed. The Delete key on Macintosh or Backspace key on Windows erases the most recently typed digit, and sends the stored number out the right outlet. The period key acts as a decimal point and causes **numkey** not to store subsequent digits until a new number is started (unless there is a float argument). Typing the Return or Enter keys on Macintosh or the Enter key on Windows sends the stored number out the left outlet and resets the number stored in **numkey** to 0, so that a new number can be typed in.

float When there is a float argument, **numkey** understands decimal points and fractional parts of a number, and sends out floats instead of ints.

numkey

Interpret numbers typed on the computer keyboard

Examples


key
numkey
0
12
The number is sent out as it is being typed
Return or Enter sends it out the left outlet
key
numkey
127
127
Type number commands in from the Mac keyboard
key
numkey
99
pgmout a 1

Recognizes all numbers typed in

See Also

key	Report key presses on the computer keyboard
keyup	Report key releases on the computer keyboard
number box	Display and output a number
Tutorial 20	Using the computer keyboard

offer

Store x,y pairs of numbers temporarily

Input

list	In left inlet: The first number is the *x* value, and the second number is the *y* value, of an *x,y* pair to be stored in **offer**. The first number must be an int; the second number may be a float, but will be Converted to int.
int	In left inlet: The number specifies the *x* value of an *x,y* pair. If a *y* value has been received in the right inlet, the two numbers are stored together in **offer**; otherwise, **offer** looks for an *x* value that matches the incoming number, sends out the corresponding *y* value, then deletes the stored pair. If there is no *x* value stored in **offer** that matches the number received, **offer** does nothing.
	In right inlet: The number specifies a *y* value to be stored in **offer**. The next *x* value (int) received in the left inlet causes the two numbers to be stored together as an *x,y* pair.
float	In right inlet: Converted to int.
clear	In left inlet: Deletes the entire contents of **offer**.

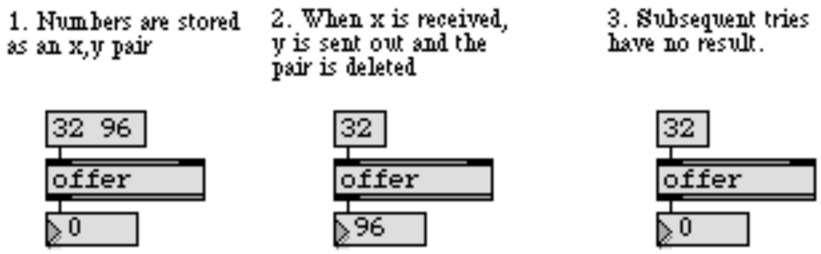
Arguments

None.

Output

int	If the number received in the left inlet matches the *x* value of an *x,y* pair stored in **offer**, the corresponding *y* value is sent out and the stored pair is deleted.

Examples


1. Numbers are stored as an x,y pair
2. When x is received, y is sent out and the pair is deleted
3. Subsequent tries have no result.
32 96
offer
0
32
offer
96
32
offer
0

A pair of numbers can be stored, then recalled a single time.

See Also

coll	Store and edit a collection of different messages
funbuff	Store x,y pairs of numbers together
table	Store and graphically edit an array of numbers

onebang

Traffic control for bang *messages*

Input

bang	In left inlet: Causes a bang to be sent out the left inlet only if a bang has been received in the right inlet since the last bang was sent out. In right inlet: Resets **onebang** to permit a bang to be sent out the next time a bang is received in the left inlet.
stop	In left inlet: Undoes the effect of a bang in the right inlet.
anything	In either inlet: Converted to bang.

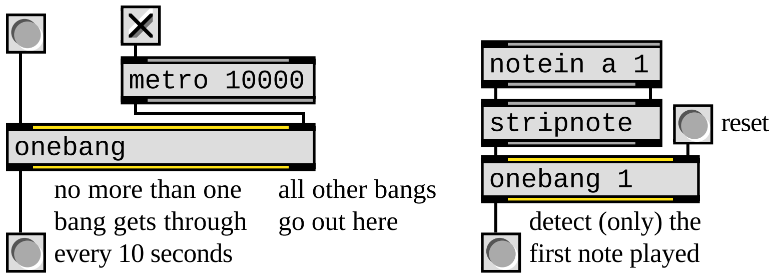
Arguments

int	Optional. A non-zero argument sets **onebang** to permit a bang to be sent out the left outlet the first time a bang is received in the left inlet.

Output

bang	When **onebang** receives a bang in its left inlet, it sends a bang out its left outlet *only* if it has received a bang in its right inlet since the last time it sent out a bang. Otherwise, it sends a bang out its right outlet.

Examples



metro 10000
onebang
no more than one bang gets through every 10 seconds
all other bangs go out here
notein a 1
stripnote
reset
onebang 1
detect (only) the first note played

Allow just one of (potentially) many bang *messages to get through*

See Also

gate	Pass the input out a specific outlet
Ggate	Pass the input out one of two outlets

onecopy

Prevent multiple copies of the same patcher from being opened

Use the **onecopy** object inside a patcher that you want to place in the extras folder for inclusion in the Extras menu. When the patcher's name is chosen using the Extras menu, its window will be brought to the front instead of opened a second time if it has already been loaded. The patch will be loaded if it is not currently open. The **onecopy** object cooperates with the Extras menu to ensure that only one copy of the patcher is opened at a time. However, opening the patcher containing a **onecopy** object by choosing Open... from the File menu will open additional copies.

Input

None.

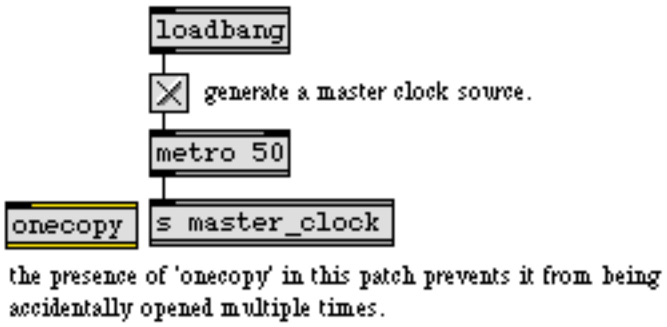
Arguments

None.

Output

None.

Example



loadbang
generate a master clock source.
metro 50
onecopy
s master_clock
the presence of 'onecopy' in this patch prevents it from being accidentally opened multiple times.

Use **onecopy** *to prevent multiple copies of the same patch from being opened from the Extras menu*

See Also

thispatcher	Send messages to a patcher
pcontrol	Open and close subwindows within a patcher

opendialog

Open a dialog to ask for a file or folder

Input

bang	Opens a standard Open Document dialog box for choosing a file.
set	The word set, followed by a four-letter symbol (e.g., TEXT, maxb) which specifies a file type, sets the **opendialog** object to search for the designated file type when opening the dialog box.
sound	Sets **opendialog** to list audio files (AIFF, Sound Designer II, NeXT/Sun, and WAV, along with some generic data file types).
types	The word types, followed by one or more four-letter type codes, determines which file types are listed by the **opendialog** object. Example type codes for files are TEXT for text files, maxb for Max binary format patcher files, and AIFF for AIFF format audio files. types with no arguments makes the object accept all file types, which is the default setting.
any symbol	One or more symbols are interpreted as one or more type codes used to determine which files are listed by the **opendialog** object.

Arguments

fold	Optional. Sets **opendialog** to choose folders instead of files.
sound	Optional. Sets **opendialog** to list audio files (AIFF, Sound Designer II, NeXT/Sun, and WAV, along with some generic data file types). The QuickTime appendix lists all the files that can be opened.
any symbol	Optional. One or more symbols set the list of file types that determine which files are listed by the **opendialog** object.

Output

symbol — Out left outlet: The absolute pathname of the file chosen by the user as a symbol. The output pathnames contain slash separators.

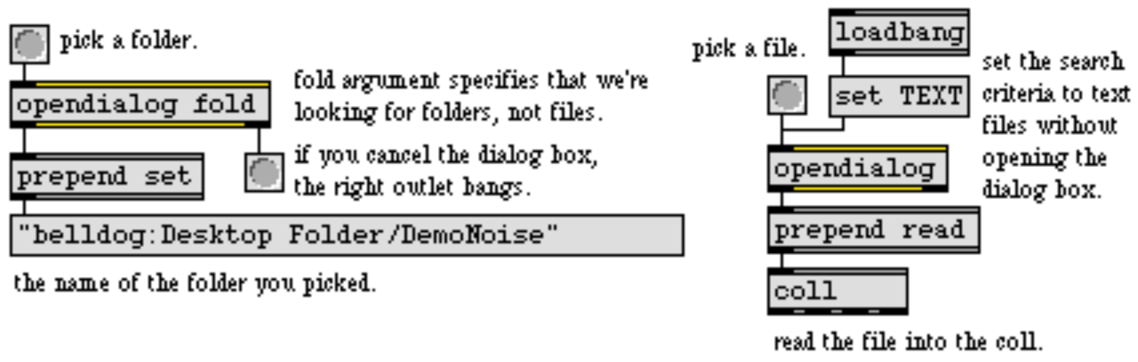
Absolute pathnames look like this:

"C:/Max Folder/extras/mystuff/mypatch.pat"

The **conformpath** object can be used to convert paths of one pathtype and/or pathstyle to another.

bang — If the dialog box is cancelled by the user, a bang message is sent out the right outlet.

Examples


pick a folder.
opendialog fold
fold argument specifies that we're looking for folders, not files.
prepend set
if you cancel the dialog box, the right outlet bangs.
"belldog:Desktop Folder/DemoNoise"
the name of the folder you picked.
pick a file.
loadbang
set TEXT
set the search criteria to text files without opening the dialog box.
opendialog
prepend read
coll
read the file into the coll.

Look for folders or a certain kind of file

See Also

conformpath	Convert paths of one pathtype and/or pathstyle to another
dialog	Open a dialog box for text entry
dropfile	Define a region for dragging and dropping a file
date	Report current date and time
filedate	Report the modification date of a file
filein	Read in a file of binary data
filepath	Report information about the current search path
folder	List the files in a specific folder
strippath	Get filename from an absolute pathname

outlet

Send messages out of a patcher

Input

(patcher) Each **outlet** object in a patch will show up as an outlet at the bottom of an object box when the patcher is used inside another patcher (as an object or a subpatch). Messages received in the **outlet** object in the subpatch will come out of corresponding outlet in the subpatch's object box in the patcher that contains it.

Inspector

A descriptive Assistance message can be assigned to an **outlet** object and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **outlet** object displays the **outlet** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

Typing in the *Describe Outlet* text area specifies the content of the Assistance message.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

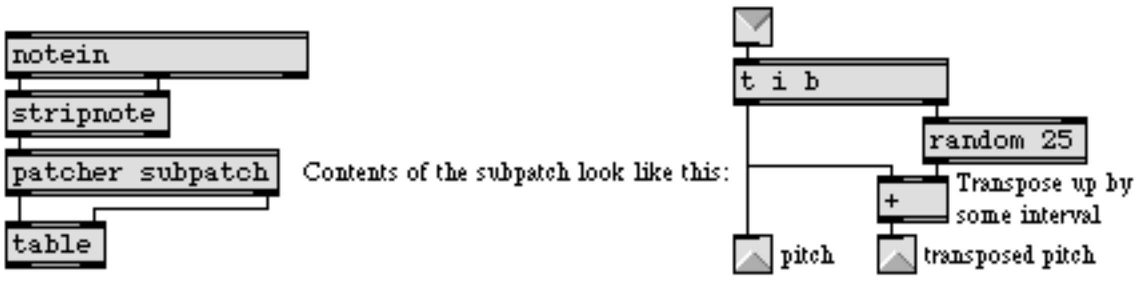
Arguments

None.

Output

anything Any messages received by **outlet** in a subpatch are sent out the outlet of that subpatch, through patch cords.

Examples



Outlets of the subpatch object correspond to the **outlet** *objects inside the subpatch*

See Also

bpatcher	Embed a visible subpatch inside a box
forward	Send remote messages to a variety of objects
inlet	Receive messages from outside a patcher
patcher	Create a subpatch within a patch
receive	Receive messages without patch cords
send	Send messages without patch cords
Tutorial 26	The **patcher** object

oval

Draw solid oval in a graphic window

Input

bang — In left inlet: Draws the oval using the current screen coordinates, drawing mode, and color.

int — In left inlet: Sets the left screen coordinate of the oval and draws the shape.

In 2nd inlet: Sets the top screen coordinate of the oval.

In 3rd inlet: Sets the right screen coordinate of the oval.

In 4th inlet: Sets the bottom screen coordinate of the oval.

In 5th inlet: Sets the drawing mode of the oval. The following are drawing mode constants; not all modes will be available on all operating systems.

Copy	0	blend	32
Or	1	addPin	33
Xor	2	addOver	34
Bic	3	subPin	35
NotCopy	4	transparent	36
NotOr	5	adMax	37
NotXor	6	subOver	38
NotBic	7	adMin	39

In 6th (right) inlet: Sets the palette index (color) of the oval according to the graphics window’s current palette. This setting has no effect when the monitor is in black and white mode.

frgb — In left inlet: The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the color of the oval the next time it is drawn.

priority — In left inlet: The word priority, followed by a number greater than 0, sets an **oval** object’s sprite priority in its graphics window. Objects with lower priority will draw behind those with a higher priority.

Arguments

any symbol — Obligatory. The first argument to **oval** must be the name of a graphics window into which the oval will be drawn. The window need not exist at the time the **oval** object is created, but the oval will not be drawn unless the name matches that of an existing and visible window.

int — Optional. Sets the initial sprite priority of the oval. If no priority is specified, the default is 3.

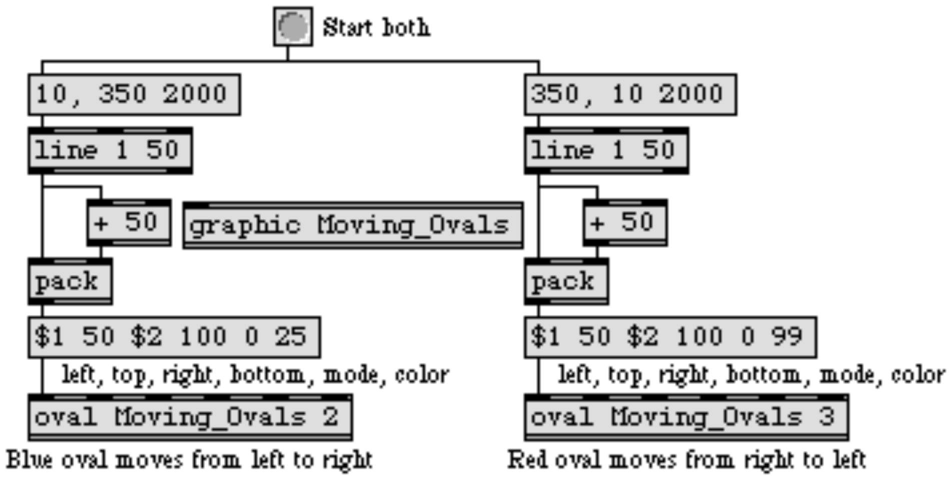
oval

Draw solid oval in a graphic window

Output

(visual)	When the **oval** object's associated graphics window is visible, and a bang message or a number is received in its left inlet, a shape is drawn in the window, and the object's previously drawn oval (if any) is erased.

Examples



*The **oval** object on the right will appear to pass in front of the one on the left when both move across the screen, since it has a higher sprite priority*

See Also

frame	Draw framed rectangle in a graphic window
graphic	Window for drawing sprite-based graphics
lcd	Draw graphics in a patcher window
rect	Draw solid rectangle in a graphic window
ring	Draw framed oval in a graphic window
Graphics	Overview of Max graphics windows and objects

Input

int	The number is stored in **pack** as an item in a list, with its position in the list corresponding to the inlet in which it was received. A number in the left inlet is stored as the first item in the list, and causes the entire list to be sent out the outlet. If the inlet in which the number is received has been initialized with a float or symbol argument, the incoming number will be converted to a float or a blank symbol, respectively.
float	The number is stored in **pack** as an item in a list, with its position in the list corresponding to the inlet in which it was received. A number in the left inlet is stored as the first item in the list, and causes the entire list to be sent out the outlet. If the inlet in which the number is received has been initialized with an int or symbol argument, the incoming number will be converted to an int or a (blank) symbol, respectively. If no argument has been typed in, float is converted to int.
bang	In left inlet: Causes **pack** to send out a list of the items currently stored.
any symbol	If the inlet in which the symbol is received has been initialized with a symbol argument, the symbol is stored in the corresponding location in **pack**. Otherwise, the symbol is converted to 0 before being stored. A symbol in the left inlet triggers output of the **pack** object's contents.
list	Any multi-item message, regardless of whether it begins with a number, is treated as a list by **pack**. The first item in the incoming list is stored in **pack** in the location that corresponds to the inlet in which it was received, and each subsequent item is stored as if it had arrived in subsequent inlets (limited by the number of inlets available). A list received in the left inlet causes the entire stored list to be sent out the outlet.
set	The word set, followed by any message, allows that message to be received by **pack** without triggering any output. Although a set message may be received in any inlet, it is only meaningful in the left inlet, which is the only triggering inlet. In any other inlet, the word set is ignored and the rest of the message is used as normal.
nth	The word nth, followed by the number of an inlet (starting at 1 for the leftmost inlet), causes the value of the item stored at that location in **pack** to be sent out the outlet.
send	In left inlet: The word send, followed by the name of a **receive** object, sends a list of the currently stored items to all **receive** objects with that name, instead of out **pack** object's outlet.

pack

Combine numbers and symbols into a list

Arguments

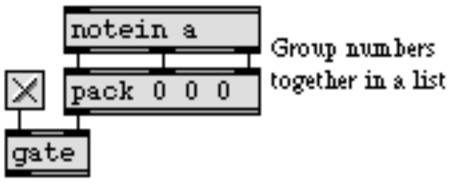
int, float, symbol — Optional. The number of inlets is determined by the number of arguments. Each argument sets an initial type and value for an item in the list stored by **pack**. If a number argument contains a decimal point, that item will be stored as a float. If the argument is a symbol, that item will be stored as a symbol. If there is no argument, there will be two inlets, and the two list items will be set to (int) 0 initially. Note: Typing a list into an object box automatically identifies it as a **pack** object, so you may omit the word **pack** from the object box, provided that you type in a list of arguments (that has at least two items and begins with a number).

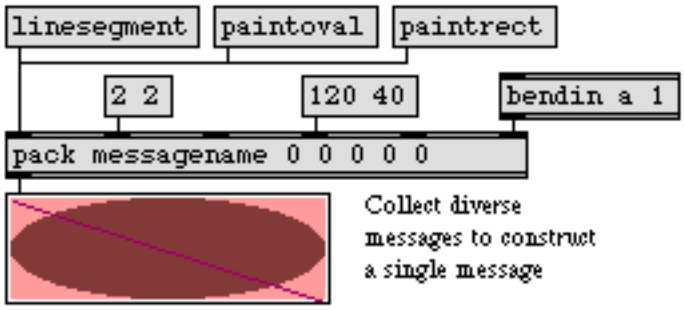
Output

list — The length of the list is determined by the number of arguments. When input is received in the left inlet, the stored list is sent out the outlet.

int, float, symbol — When the nth message is received, the value of the specified item is sent out.

Examples




linesegment
paintoval
paintrect
2 2
120 40
bendin a 1
pack messagename 0 0 0 0 0
Collect diverse messages to construct a single message

Numbers and symbols may be mixed as needed in **pack**

See Also

bondo	Synchronize a group of messages
buddy	Synchronize arriving data, output them together
match	Look for a series of numbers, output it as a list
swap	Reverse the sequential order of two numbers
thresh	Combine numbers into a list, when received close together
unpack	Break a list up into individual messages
zl	Multi-purpose list processor
Tutorial 30	Number groups

panel

Colored background area

The **panel** object lets you create rectangular background panels for use in creating user interfaces. You can also create rectangles with rounded corners and shading which can also be used as buttons when used in conjunction with **ubutton** object.

Input

border	The word border, followed by a number, sets the size, in pixels of the **panel** object's border. The default is 1.
brgb	The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the color (Background) of the **panel** object. The default value is gray (brgb 192 192 192).
frgb	The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the border of the **panel** object. The default value is black (frgb 0 0 0).
rounded	The word rounded, followed by a number, sets the size, in pixels of the rounding of the **panel** object's corners. The default is 0 (no rounding).
shadow	The word rounded, followed by a positive or negative number, sets the size, in pixels for a "shadow" effect for the **panel** object. Positive numbers create a "raised" shadow effect, and negative numbers created a "recessed" effect. The default is 0 (no shadow).
size	The word size, followed by two numbers, specifies the width and height, in pixels, of the **panel** object. The default panel size has a width of 69 and a height of 57.

Inspector

The behavior of a **panel** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **panel** object displays the **panel** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The *Width* and *Height* number boxes are used to set the size of the panel. The default panel size has a width of 69 and a height of 57. *Border Size* specifies the width, in pixels of the panel border. The default is 1. Entering a value in the *Shadow Size* number box sets the size of the panel's shadow. The default is 0 (no shadow). The number, of pixels, worth of rounding for the panel is specified by entering a number into the *Rounded Corners* box. The default is 0 (no rounding).

The *Color* option lets you use a swatch color picker or RGB values used to set the border color and the frame color. *Frame* sets the color for the border of the **panel** object (default 0 0 0), and *Background* sets the color for the panel (default 192 192 192).

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

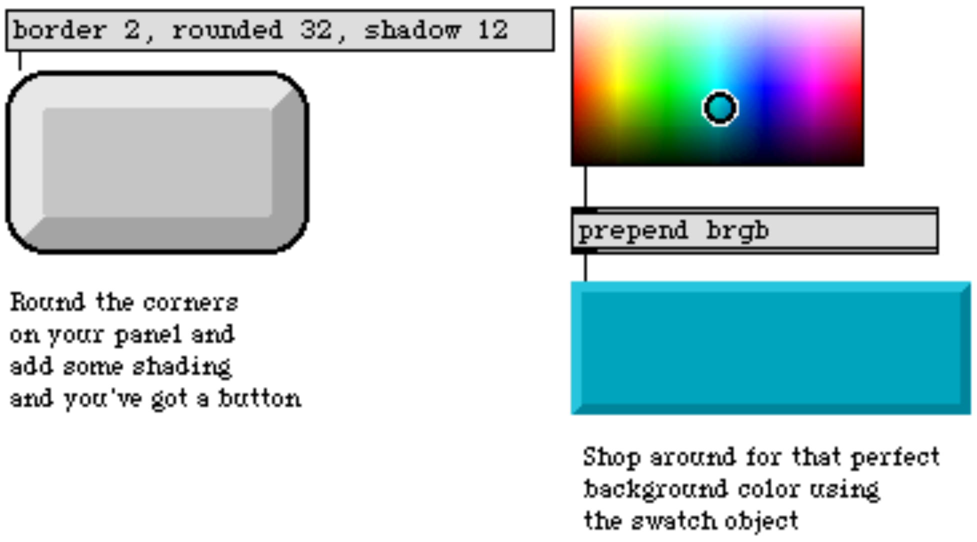
Arguments

None.

Output

None.

Examples



```
border 2, rounded 32, shadow 12

Round the corners
on your panel and
add some shading
and you've got a button

prepend brgb

Shop around for that perfect
background color using
the swatch object
```

See Also

fpic	Display a picture from a graphics file
lcd	Draw graphics in a patcher window
pict	Draw picture in a graphic window
ubutton	Transparent button, sends a bang

past

Report when input increases beyond a certain number

Input

list	The numbers in the list are compared to the arguments. If all of the numbers in the list are *greater than or equal to* the corresponding arguments, a bang is sent out the outlet. Before a bang is *sent again*, however, **past** must receive a clear message, or must receive another list in which the number that equaled or exceeded its argument goes back below (is less than) its argument.
int or float	If there is only one argument, and the input is *greater than or equal to* it, and the previous input was *not* greater than or equal to it, **past** sends a bang out the outlet.
clear	Causes **past** to forget previously received input, readying it to send a bang messageagain.
set	The word set, followed by one or more numbers, sets the numbers which must be equaled or exceeded by the numbers received in the **past** object's inlet.

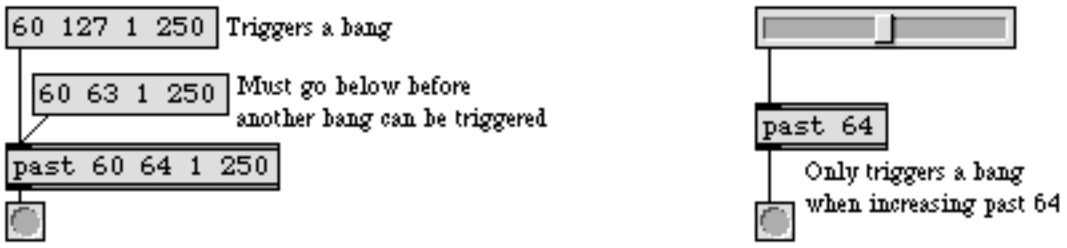
Arguments

list	Sets the numbers which must be equaled or exceeded by the numbers received in the inlet.
int	Sets a single number which must be equaled or exceeded by the number received in the inlet.

Output

bang	If all of the arguments are equaled or exceeded by the numbers received in the inlet, **past** sends out a bang. Otherwise, **past** does nothing. A bang is sent only as a number *increases* past its threshold. Once the threshold has been passed, the number must go below the threshold again, then increase past it, before another bang will be sent.

Examples



60 127 1 250
Triggers a bang
60 63 1 250
Must go below before another bang can be triggered
past 60 64 1 250
past 64
Only triggers a bang when increasing past 64

Send out bang *only when the input goes past the threshold in an upward direction*

See Also

maximum	Output the greatest in a list of numbers
peak	If a number is greater than previous numbers, output it
>	*Is greater than*, comparison of two numbers

patcher / p

Create a subpatch within a patch

Input

anything	The number of inlets in a **patcher** object is determined by the number of **inlet** objects contained within its subpatch window.

Arguments

any symbol(s)	Optional. The subpatch can be given a name by the argument, so that its name appears in the title bar of the subpatch window. The name in the title bar of the subpatch window is displayed in brackets to indicate that it is part of another file. If there is no argument typed in, the subpatch window is named [sub patch]. Different patcher objects that share the same name are still distinct subpatches, and do not share the same contents.

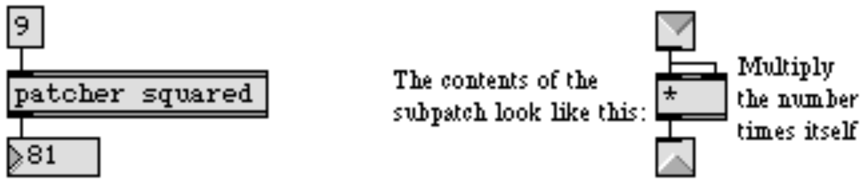
Output

anything The number of outlets a patcher object has is determined by the number of **outlet** objects contained within the subpatch window. Output can also be sent via **send** and **value** objects contained in the subpatch. The actual messages sent out of a patcher object depend on the contents of the subpatch.

When a patcher object is first created, the subpatch window is automatically opened for editing. To view or edit the contents of a patcher object (or any subpatch object) later on, double-click on the object when the patcher window is locked.

All the objects in a subpatch of a patcher object are saved as part of the patcher which contains the object.

Examples


9
patcher squared
81
The contents of the subpatch look like this:
*
Multiply the number times itself

A patch can be contained (and saved) as part of another patch

See Also

bpatcher	Embed a visible subpatch inside a box
inlet	Receive messages from outside a patcher
outlet	Send messages out of a patcher
pcontrol	Open and close subwindows within a patcher
thispatcher	Send messages to a patcher
Tutorial 26	The **patcher** object

pcontrol

Open and close subwindows within a patcher

Input

open Opens the patcher window of any subpatches or patcher objects connected to the **pcontrol** object's outlet.

close Closes the patcher window of any subpatches or patcher objects connected to the **pcontrol** object's outlet.

enable The word enable, followed by any number other than 0, enables the MIDI objects contained in the subpatches or patcher objects connected to the **pcontrol** object's outlet. A message of enable 0 *disables* the MIDI objects in those subpatches.

If a second non-zero numerical argument is added, the enable message will disable/enable the patcher *and its subpatchers*. The enable message also affects the enabling/disabling of MSP audio processing (in addition to MIDI) within the selected patch.

load The word load, followed by the name of a patcher file, opens that file if it can be found in Max's search path. The file name may optionally be followed by up to nine numbers and/or symbols, which will be substituted for the appropriate changeable # arguments (#1 to #9) in the patch being opened.

shroud The word shroud, followed by the name of a patcher file, opens that file *but does not show its window.* (Use this message with care, since having patchers open but invisible can potentially lead to some disconcerting results.)

help The word help, followed by a symbol, opens a help file in Max's max-help folder with the name of the symbol followed by .help.

Arguments

None.

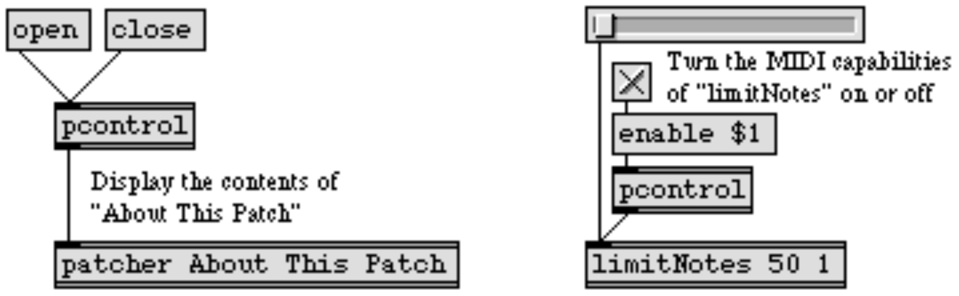
Output

Any subpatches or patcher objects connected to the **pcontrol** object's outlet can have their patcher window opened or closed, or MIDI enabled/disabled, when the appropriate message is received in the inlet of **pcontrol**.

pcontrol

Open and close subwindows within a patcher

Examples



open
close
pcontrol
Display the contents of "About This Patch"
patcher About This Patch
Turn the MIDI capabilities of "limitNotes" on or off
enable $1
pcontrol
limitNotes 50 1

Show/hide a subpatch window, or enable/disable its MIDI objects

See Also

bpatcher	Embed a visible subpatch inside a box
inlet	Receive messages from outside a patcher
patcher	Create a subpatch within a patch
thispatcher	Send messages to a patcher
Tutorial 40	Automatic actions

peak

If a number is greater than previous numbers, output it

Input

int In left inlet: If the input is greater than the value currently stored in **peak**, it is stored as the new peak value and is sent out.

In right inlet: The number is stored in **peak** as the new peak value, and is sent out.

float Converted to int.

list In left inlet: The second number is stored as the new peak value and is sent out, then the first number is received in the left inlet.

bang In left inlet: Sends the currently stored peak value out the left outlet.

Arguments

None. The initial value stored in **peak** is 0.

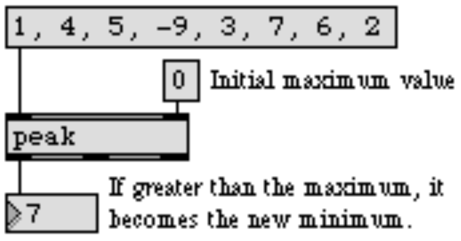
Output

int Out left outlet: New peak values are sent out. (A number received in the right inlet is always the new peak value.)

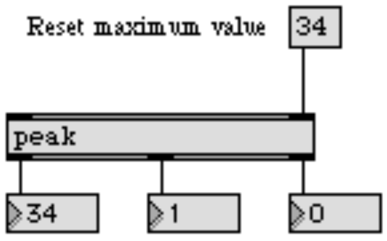
Out middle outlet: If the number received is a new peak value, the output is 1. If the number received in the left inlet is *not* a new peak value, the output is 0.

Out right outlet: If the number received is a new peak value, the output is 0. If the number received in the left inlet is *not* a new peak value, the output is 1.

Examples



Find the greatest in a series of numbers



A number in the right inlet always sets a new peak

peak

If a number is greater than previous numbers, output it

See Also

maximum	Output the greatest in a list of numbers
past	Report when input increases beyond a certain number
trough	If a number is less than previous numbers, output it
>	*Is greater than*, comparison of two numbers

pgmin

Output received MIDI program change values

Input

(MIDI)	**pgmin** receives its input from a MIDI program change message received from a MIDI input device.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming program change messages. The word port is optional and may be omitted.
int	The number is treated as if it were an incoming MIDI program change value. If there is a right outlet, 0 is sent out in lieu of a MIDI channel number. The program number plus 1 is sent out the left outlet, and is not limited in the range 1 to 128.
(mouse)	Double-clicking on a **pgmin** object shows a pop-up menu for choosing a MIDI port or device.

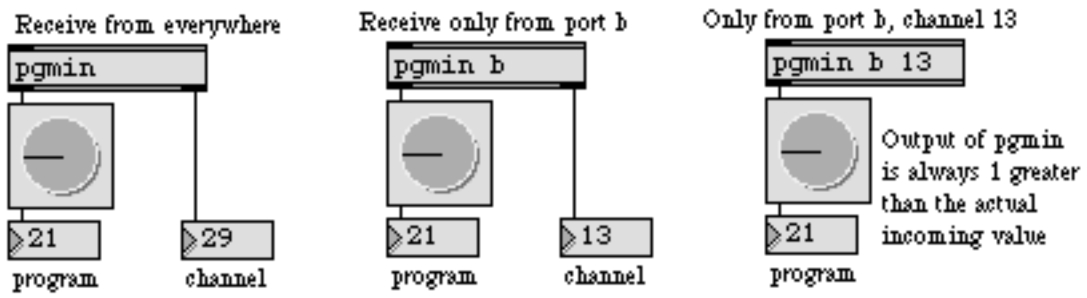
Arguments

a-z	Optional. Specifies the port from which to receive incoming program change messages. If there is no argument, **pgmin** receives from all channels on all ports.
(MIDI name)	Optional. The name of a MIDI input device may be used as the first argument to specify the port.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to receive program change messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the MIDI Setup dialog.

Output

int	If a specific channel number is included in the argument, there is only one outlet. The output is the incoming program number on the specified channel and port. Note: The pgmin object always adds 1 to the incoming program number. Thus, an incoming program change value of 32 will come out the outlet of **pgmin** as 33. If there is no channel number specified by the argument, **pgmin** will have a second outlet, on the right, which will output the channel number of the incoming program change message.

Examples



Program changes can be received from everywhere, a specific port, or a specific port and channel

See Also

midiin	Output received raw MIDI data
pgmout	Transmit MIDI program change messages
Tutorial 16	More MIDI ins and outs
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified

pgmout

Transmit MIDI program change messages

Input

int	In left inlet: The number has 1 subtracted from it and then is transmitted as a program change value on the specified channel and port. Numbers are limited between 1 and 128, and are sent out as program changes 0 to 127.
	In right inlet: The number is stored as the channel number on which to transmit the program change messages.
float	Converted to int.
list	In left inlet: The first number is the program number +1, and the second number is the channel, of a MIDI program change message, transmitted on the specified channel and port.
enable	The message enable 0 disables the object, causing it not to transmit MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI output port or device, specifies the port used to transmit the MIDI messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **pgmout** object shows a pop-up menu for choosing a MIDI port or device.

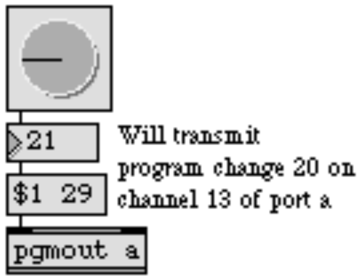
Arguments

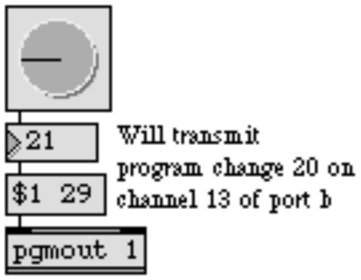
a-z	Optional. Specifies the port for transmitting MIDI program change messages. When a letter argument is present, channel numbers greater than 16 received in the right inlet will be *wrapped around* to stay within the 1-16 range. If there is no argument, **pgmout** initially transmits out port a, on MIDI channel 1.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to transmit program change messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
(MIDI name)	Optional. The name of a MIDI output device may be used as the first argument to specify the port.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

Output

(MIDI)	There are no outlets. The output is a MIDI program change message transmitted directly to the object's MIDI output port.

Examples





Letter argument transmits to only one port. Otherwise, number specifies both port and channel

See Also

midiout	Transmit raw MIDI data
pgmin	Output received MIDI program change values
Tutorial 16	More MIDI ins and outs
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified

pict

Draw picture in a graphic window

Input

bang — Draws the picture stored in the **pict** object if its associated graphics window is visible.

clear — Erases the picture drawn in the graphics window.

int — In left inlet: A nonzero number draws the picture in its associated graphics window if that window is visible. 0 erases the picture.

In middle inlet: Sets the left edge of the picture, in pixels, relative to the left edge of the graphics window (effective the next time the picture is drawn).

In right inlet: Sets the top edge of the picture, in pixels, relative to the top edge of the graphics window's drawing area (effective the next time the picture is drawn).

priority — The word priority, followed by a number greater than or equal to 0, sets the object's *sprite priority* to that number. Refer to the *Graphics* section of the **Tutorials and Topics** manual for a discussion of sprite priorities.

Arguments

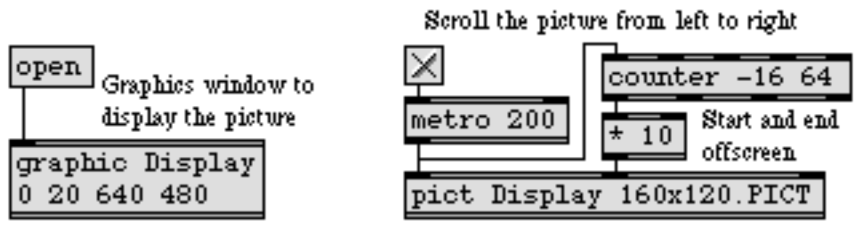
symbol — Obligatory. The first argument to **pict** must be the name of a **graphic** object whose window will be used to draw the picture. The second argument must be the name of a Quicktime PICT file which will be loaded when the object is created. PICT files have .pct filename extensions on Windows.

int — Optional. Following the window name and file name, a number greater than or equal to 0 sets the initial sprite priority. The default priority is 0, which means the picture will be drawn behind all other objects. Following the priority number, the next two arguments specify the left and top offsets of the image, in pixels, relative to the top left corner of the graphics window's drawing area.

Output

(visual) — When the **pict** object's associated graphics window is visible, and a bang message or a nonzero int is received in its inlet, the stored picture is drawn in the window.

Examples


Scroll the picture from left to right
open
Graphics window to display the picture
graphic Display 0 20 640 480
metro 200
counter -16 64
* 10
Start and end offscreen
pict Display 160x120.PICT

Picture can be displayed or moved around in the graphics window

See Also

frame	Draw framed rectangle in a graphic window
graphic	Window for drawing sprite-based graphics
lcd	Draw graphics in a patcher window
oval	Draw solid oval in a graphic window
rect	Draw solid rectangle in a graphic window
ring	Draw framed oval in a graphic window
Graphics	Overview of Max graphics windows and objects
Tutorial 42	Graphics

pictctrl

Picture-based control

The **pictctrl** object is a user interface object for creating buttons, switches, knobs, and other controls. It can open PICT files and, if QuickTime Version 3.0 or later is installed, other picture file formats that are listed in the QuickTime appendix.Since the **pictctrl** object uses images from a picture file for its appearance, you can create controls with whatever appearance you desire.

Note: The **pictctrl** object requires that QuickTime be installed on your system to open any files other than PICT files. If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.

Input

int	Sets the value of the button or knob set by the control, and sends the current value out the outlet. In button and toggle mode, the value must be either 0 or 1. In dial mode, the range of values is determined by **pictctrl** object's *Range* attribute.
set	The word set, followed by a number, sets the value of the button or knob to that number, without triggering output.
bang	Sends the current value of the **pictctrl** to the outlet.
clickincrement	The word clickincrement, followed by a nonzero value, sets the output value to increment by 1 each time the object is clicked (Click to Increment mode). Any movement of the mouse after clicking is ignored. When the uppermost value is reached, the value returns to zero with the next click. All other mouse tracking modes are disabled. clickincrement 0 disables Click to Increment mode.
clickedimage	The word clickedimage, followed by a nonzero value, tells the **pictctrl** object to use an alternate set of image frames in your picture file to give the dial a different appearance when the user clicks on it and drags the mouse pointer. clickedimage 0 disables this feature.
picture	The word picture, followed by a symbol that specifies a filename, designates the picture file that the **pictctrl** object will use for the control's button or dial file. The symbol used as a filename must either be the name of a file in Max's current search path, or an absolute pathname for the file (e.g."MyDisk:/Documents/UI Pictures/CoolKnob.pct"). The word picture by itself puts up a standard Open Document dialog box and displays the common graphics files supported by QuickTime.
active	The word active, followed by a 0 or 1, toggles mouse control of the **pictctrl** object. The default is 1 (enabled). If a separate set of inactive images is present in the **pictctrl** object's picture file and if the inactive images attribute is set, the active message will also change the appearance of the control.
inactiveimage	The word inactiveimage, followed by a nonzero value, tells the **pictctrl** object that your picture file has an additional row of images for its inactive state. The default is 0 (no inactive state).

imagemask	The word imagemask, followed by a nonzero value, tells the **pictctrl** object that your picture file has an image mask. The default is 0 (no image mask).
tracking	The word tracking, followed by a 0 or 1, toggles live tracking. If live tracking is on, the **pictctrl** object will change its state if the mouse moves in and out of the rectangular border of the object with the mouse button held down. tracking 0 disables live tracking
range	The word range, followed by a number, sets the range of the **pictctrl** object when it is in dial mode. The default value is 128.
offset	The word offset, followed by a number, sets an offset value. When **pictctrl** is in dial mode, the offset value is added to the object's value before being sent out the outlet. The default offset value is 0.
multiplier	The word multiplier, followed by a number, specifies a multiplier value. When **pictctrl** is in dial mode, the object's value is multiplied by this number before being sent out the outlet. The multiplication happens before the addition of the Offset value. The default multiplier value is 1.
frames	The word frames, followed by a number, specifies the number of images (columns) in the picture file. The number of frames does not have to be the same as the range of the control; the **pictctrl** object will use the nearest image for any given value.
trackhorizontal	The word trackhorizontal, followed by a nonzero value, sets the **pictctrl** object to respond when you click on it and drag the mouse horizontally; moving the mouse to the right increases the object's value, and moving it to the left decreases the value. Enabling this mode of operation disables the *Circular Tracking* and *Click to Increment* modes (see the clickincrement and trackcircular messages).
trackvertical	The word trackvertical, followed by a nonzero value, sets the **pictctrl** object to respond when you click on it and drag the mouse vertically; moving the mouse up increases the object's value, and moving it down decreases the value. Enabling this mode of operation disables the *Circular Tracking* and *Click to Increment* modes (see the clickincrement and trackcircular messages).
trackcircular	The word trackcircular, followed by a nonzero value, sets the **pictctrl** object to respond when you click on it and drag the mouse in a circular arc relative to the control's center (*Circular Tracking* mode). Moving the mouse clockwise increases the control's value, and moving it counterclockwise decreases its value. Enabling circular tracking disables all other tracking modes. trackcircular 0 disables circular tracking.
ratio	The word ratio, followed by a number, specifies how many pixels the mouse pointer must move before the value of the dial changes by one increment. If the **pictctrl** object is using *Circular Tracking*, the ratio message specifies how many

degrees the cursor must move, relative to the center of the object, to increase the value by one.

Inspector

The behavior of a **pictctrl** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **pictctrl** object displays the **pictctrl** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

Some of the **pictctrl** object's attributes are associated with one of the three modes of this object—*Button Mode*, *Toggle Mode*, and *Dial Mode*. The **pictctrl** Inspector lets you set the following attributes:

Button mode imitates the behavior of simple buttons in graphical user interfaces, such as the "OK" and "Cancel" buttons found in dialog boxes. In this mode, the **pictctrl** object outputs a 1 when the user clicks on the object, and a 0 when the user either moves the mouse off of the object or releases the mouse button. Button mode is also useful for display objects, such as simulated LEDs and status indicators.

Toggle mode is similar to button mode, except that the object changes state from 0 to 1 (or 1 to 0) with every mouse click. Toggle mode imitates the behavior of check boxes.

Checking *Live Tracking* can only be done if you're using the **pictctrl** object's button mode. If this checkbox is checked, **pictctrl** will change state if the mouse moves in and out of the rectangular border of the object with the mouse button held down.

Dial mode can be used to create controls that act like knobs, or any other control that has more than two distinct values. (You could use dial mode to create sliders, but the **pictslider** object is better suited to this task.) Dial mode lets you set a range, offset, and multiplier for its values, just as with Max's **hslider**, **uslider**, and **dial** objects. When you click on the object and drag, its value changes. **pictctrl** can track either horizontal and/or vertical cursor motion, or circular motion. ignoring subsequent drag motions. When using dial mode you must specify the number of image frames that are in the picture file you're using (see below). The number of images does not have to be the same as the range of values. For example, a knob could have a range of 128 but only 30 distinct images. There is little reason to create a control with more image frames than its range, since manipulating the control could change its appearance without causing any output.

When using dial mode you must specify the number of image frames that are in the picture file you're using (see below). The number of images does not have to be the same as the range of values. For example, a knob could have a range of 128 but only 30 distinct images. There is little reason to create a control with more

image frames than its range, since manipulating the control could change its appearance without causing any output.

When the **pictctrl** object is in dial mode, you can specify a *Range* for the object which will automatically limit numbers received in the inlet to between 0 and the number 1 less than the specified range, a *Multiplier*—a number by which all numbers will be multiplied before being sent out—and an *Offset*—which will be added to the number, after multiplication. The default object has a range of 128, a multiplier of 1, and an offset of 0.

The *Image Frames* box lets you specify the number of distinct images (columns) in the picture file. The number of frames does not have to be the same as the range of the control; **pictctrl** will use the nearest image for any given value.

If *Horizontal Tracking* or *Vertical Tracking* is checked, the **pictctrl** object will respond when you click on it and drag the mouse in the corresponding direction. Dragging the mouse to the right and/or up increases the **pictctrl** object's value; dragging it left and/or down decreases its value. Enabling either of these attributes disables the Circular Tracking and Click to Increment modes (see below).

If *Circular Tracking* is checked, the control will respond when you click on it and drag the mouse in a circular arc relative to the control's center. Dragging the mouse clockwise increases the control's value; dragging it counterclockwise decreases its value. Enabling Circular Tracking disables all other tracking modes.

If *Click to Increment* is checked, the control's value increases by one every time it is clicked. Subsequent dragging motions are ignored. When the uppermost value is reached, the value returns to zero with the next click. Enabling Click to Increment disables all other tracking modes.

If *Clicked Images* is checked, **pictctrl** uses an alternate set of image frames in your picture file to give the dial a different appearance when the user clicks on it and drags the mouse pointer.

The *Tracking Ratio* attribute specifies how many pixels the mouse pointer must move before the value of the dial changes by one increment. For the circular tracking mode, the tracking ratio specifies how many degrees the cursor must move, relative to the center of the object, to increase the value by one.

The *Has Inactive Images* and *Image Masks* checkboxes specify that your picture file has additional rows of images for its inactive state, and whether it has image masks.

The *Picture File* option lets you choose a picture file for the **pictctrl** object's knob by clicking on the Open button. The current file's name appears in the text box to the left of the button. You can also choose a file by typing its name in this box, or by dragging the file's icon from the Finder into this box.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

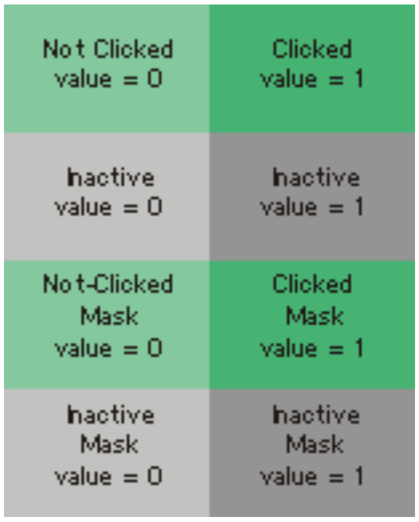
Arguments

None.

Picture File Format

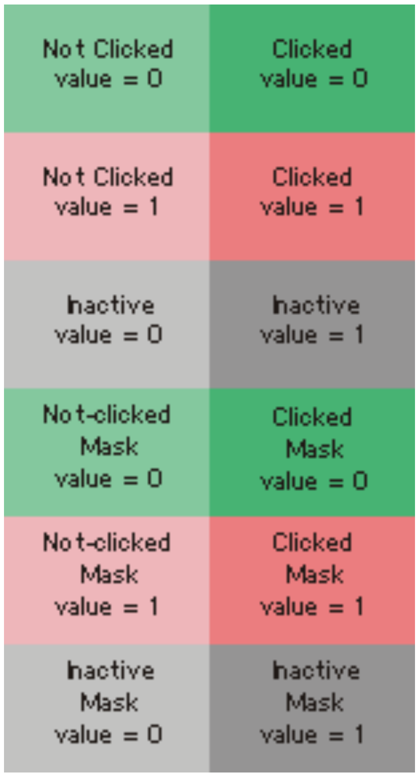
When you create a new **pictctrl** object in a patcher window, it has no associated picture file. Use the Open button in the inspector to choose a picture file for the control. It can open PICT files and, if QuickTime Version 3.0 or later is installed, other picture file formats that are listed in the QuickTime appendix. The layout of the picture in the file varies depending on which mode of operation the **pictctrl** uses. All three modes require that the pictures be made up of a grid of images, in which all images have the same width and height.

Button mode has the simplest layout:



The first row of images is mandatory: these two images are used for the idle and clicked states (values zero and one, respectively) of the button. The next row of images, if present, is used for the control when it is in its inactive state. The next rows contain the masks for the top row of images, and the inactive images if present.

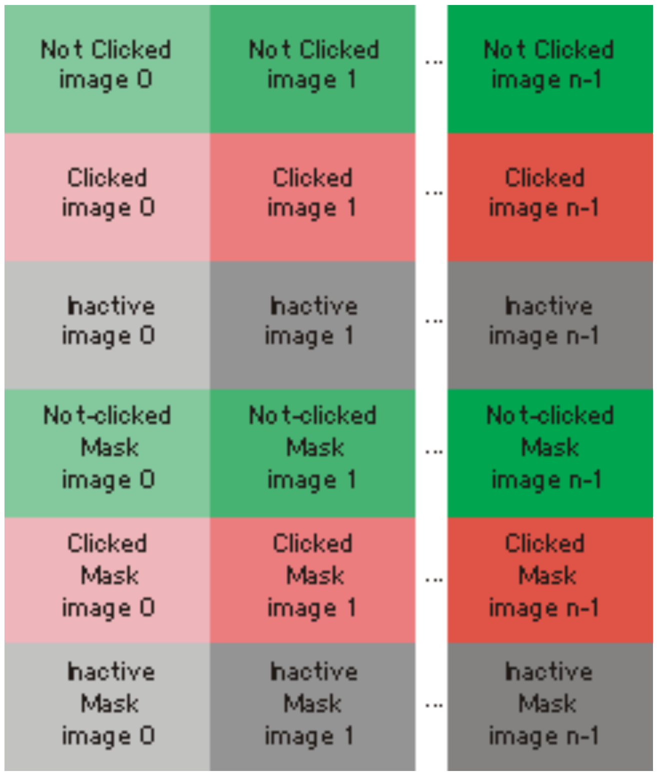
Toggle mode has a similar layout:



Not Clicked value = 0	Clicked value = 0
Not Clicked value = 1	Clicked value = 1
Inactive value = 0	Inactive value = 1
Not-clicked Mask value = 0	Clicked Mask value = 0
Not-clicked Mask value = 1	Clicked Mask value = 1
Inactive Mask value = 0	Inactive Mask value = 1

In this mode, the top two rows are mandatory. The first row of images are used when the control's value is zero, the next row when its value is one. The third row is optional; it is used for the control when it is in its inactive state. (Note that there are no "clicked" images for the inactive state, since when inactive, the control ignores mouse clicks.) The next rows contain masks for the images.

The *Dial mode* layout varies in size depending on how many image frames it has, which must be the same as the Image Frames parameter as set in the inspector:

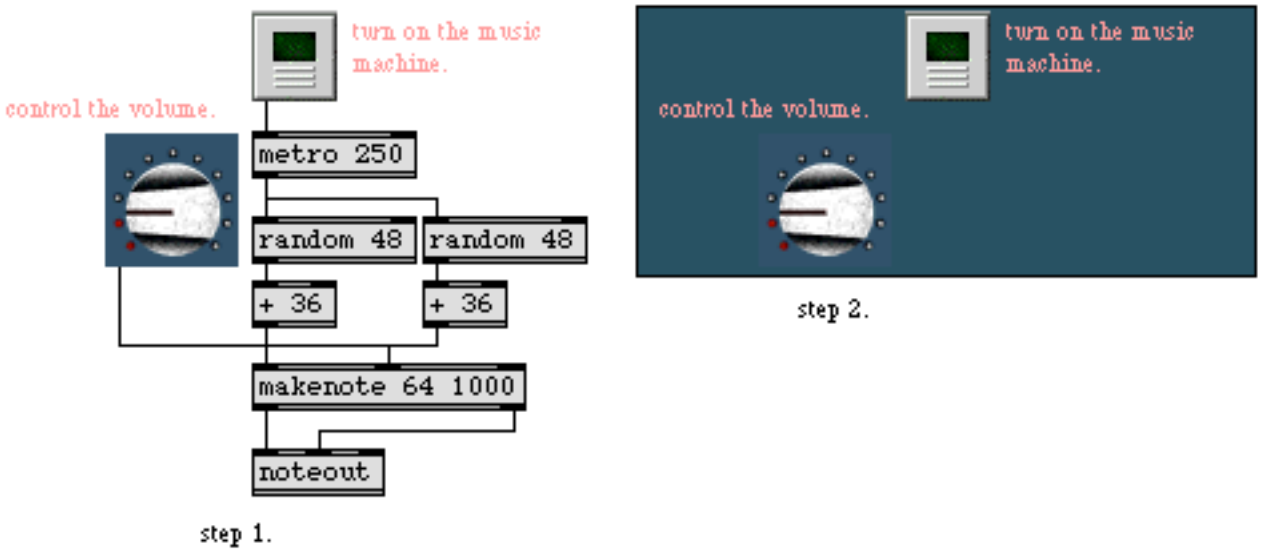


The first row of images is mandatory: one image for each visually distinct state of the control. Dials need as many picts as you wish them to have visible states. Note that dials can receive and send a larger range of values than are represented by picts (e.g. your dial can have a range of 128 even if you only use eight pict frames to represent the range of the dial). The next row of images is optional, and is used when the user is clicking and dragging on the object to change its value. The next row is also optional; (Note that there are no "clicked" images for the inactive state, since when inactive, the control ignores mouse clicks.) The following rows contain masks for the images.

Output

int — The current value of the **pictctrl** object. In toggle and button modes this will be a 0 or a 1. In dial mode, this value is specified by the range, offset, and multiplier that you set in the Inspector window.

Examples



Create customized controls to create a more attractive user interface

See Also

dial	Output numbers by moving a dial onscreen
hslider	Output numbers by moving a slider onscreen
kslider	Output numbers from a keyboard onscreen
matrixctrl	Matrix-style switch control
pictslider	Picture-based slider
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
ubutton	Transparent button, sends a bang
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

pictslider

Picture-based slider control

The **pictslider** object is a slider control that uses pictures in external files for its appearance. It uses two pictures—one for the "knob" (the part that you move with the mouse, corresponding to the part of a physical slider that you move with your fingers) and one for the background over which the knob moves. The **pictslider** object has default pictures that are used if you do not want to supply pictures of your own, but its intended use is creating controls with customized appearances.

You can use the **pictslider** object to create horizontal or vertical sliders, as well as two-dimensional controllers (virtual trackpads or joysticks).

Note: The **pictslider** object requires that QuickTime be installed on your system to open any files other than PICT files. If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.

Input

bang	In left inlet: Sends the current values of the **pictslider** to its outlets. The horizontal value is sent out the left outlet; the vertical value out its right outlet.
int	In left inlet: sets the **pictslider** object's horizontal value. The value is also sent out the left outlet, and the **pictslider** object's current vertical value is sent out the right outlet. In right inlet: sets the **pictslider** object's vertical value. The value is also sent out the right outlet, and the control's current horizontal value is sent out the left outlet.
float	Converted to int.
list	In left inlet: A list of two numbers sent to the left inlet sets the **pictslider** object's horizontal value to the first number and its vertical value to the second. The two values are sent out the left and right outlets.
active	In left inlet: The word active, followed by a 0 or 1, toggles mouse control of the **pictslider** object. The default is 1 (enabled). If a separate set of inactive images is present in the pictslider object's graphics file and if the inactive images attribute is set, the active message will also change the appearance of the control.
bkgnddrag	In left inlet: The word bkgnddrag, followed by a 0 or 1, toggles background drag mode for the **pictslider** object. When this mode is enabled, clicking and dragging *anywhere* in the background area of the slider will move the knob; the knob will move relative to the motion of the mouse, just as if you had clicked in the knob itself. The message bkgnddrag 0 disables this mode. You must also uncheck the *KnobJumps to Click Location* checkbox in the **pictslider** object's inspector or send the object a jump 0 message to enable this mode.
bkgndpicture	The word bkgndpicture, followed by a symbol that specifies a filename, designates the graphics file that the **pictslider** object will use for the control's background image. The symbol used as a filename must either be the name of a file in Max's

	current search path, or an absolute pathname for the file (e.g. “MyDisk:/Documents/UI Pictures/CoolBkgnd.pct”).
bkgndsize	In left inlet: The word bkgndsize, followed by a nonzero value, tells the **pictslider** object to change the size of the object to match the size of the background picture. After receiving this message, the object’s size cannot be changed. bkgndsize 0 allows the control to be resized in the usual manner by dragging its lower-right corner.
bottommargin	In left inlet: The word bottommargin, Followed by an int greater than or equal to zero, sets the bottom margin, in pixels, for the **pictslider**. The margin reduces the area in which the knob moves; if a margin is zero, the knob can move all the way to the bottom of the slider.
bottomvalue	In left inlet: The word bottomvalue, followed by an int, sets the values emitted by the **pictslider** object when the knob is moved as far as possible to the bottom. The message bottomvalue 100 will cause the control to send 100 out of its left outlet when the knob is moved all the way to the bottom.
clickedimage	In left inlet: The word clickedimage, followed by a nonzero value, specifies that the graphics file used by the **pictslider** object contains an additional image to be displayed when the control is clicked.
horizontaltracking	In left inlet: The word horizontaltracking, followed by a float, sets the horizontal tracking ratio for movements of the **pictslider** object’s knob. The default value is 1.0. Values greater than one cause the knob to move more quickly when dragged; values less than one cause it to move more slowly.
imagemask	In left inlet: The word imagemask, followed by a nonzero value, specifies that the graphics file used by the **pictslider** object contains image masks.
inactiveimage	In left inlet: The word inactiveimage, followed by a nonzero value, specifies that the graphics file used by the **pictslider** object contains additional images for the object’s inactive state.
invisiblebkgnd	In left inlet: The word invisiblebkgnd, followed by a nonzero value, tells the **pictslider** object to not draw any background image. The knob will appear to float above any objects underneath it.
jump	In left inlet: The word jump, followed by a nonzero value, makes **pictslider** move the knob to the position of the cursor if you click in the object outside of the knob. jump 0 disables this behavior; you must click in the knob itself to move it.
knobpicture	In left inlet: The word knobpicture, followed by a symbol that specifies a filename, designates the graphics file that the **pictslider** object will use for the control’s knob file. The symbol used as a filename must either be the name of a file in Max’s current search path, or an absolute pathname for the file (e.g. “MyDisk:/Documents/UI Pictures/CoolKnob.pct”). The word knobpicture by itself puts up a standard Open

Document dialog box and displays the common graphics files supported by QuickTime.

leftmargin — In left inlet: The word leftmargin, followed by an int greater than or equal to zero, sets the left margin, in pixels, for the **pictslider**. The margin reduces the area in which the knob moves; if a margin is zero, the knob can move all the way to the left of the slider.

leftvalue — The word leftvalue, followed by an int, sets the values emitted by the **pictslider** object when the knob is moved as far as possible to the left. The message leftvalue 100 will cause the control to send 100 out of its left outlet when the knob is moved all the way to the left.

movehorizontal — In left inlet: The word movehorizontal, followed by a nonzero value, allows the knob to change when the mouse is moved horizontally. The message movehorizontal 0 prevents the knob from moving when the mouse is moved horizontally.

movevertical — In left inlet: The word movevertical, followed by a nonzero value, allows the knob to change when the mouse is moved vertically. The message movevertical 0 prevents the knob from moving when the mouse is moved vertically.

rightmargin — In left inlet: The word rightmargin, followed by an int greater than or equal to zero, sets the right margin, in pixels, for the **pictslider**. The margin reduces the area in which the knob moves; if a margin is zero, the knob can move all the way to the right of the slider.

rightvalue — In left inlet: The word rightvalue, followed by an int, sets the values emitted by the **pictslider** object when the knob is moved as far as possible to the right. The message rightvalue 100 will cause the control to send 100 out of its left outlet when the knob is moved all the way to the right.

scaleknob — In left inlet: The word scaleknob, followed by a nonzero value, tells the **pictslider** object to stretch or shrink the knob when you change the size of the entire object. scaleknob 0 will result in the knob always being drawn at its original size.

set — In left inlet: The word set, followed by a number, sets the pictcslider object's horizontal value but does not send the value out its left outlet.The word set, followed by two numbers, sets the **pictslider** object's horizontal value to the first number and its vertical value to the to the second number, but does not send the values out its outlets.

In right inlet: The word set, followed by a number, sets the **pictslider** object's vertical value, but does not send the value out its right outlet.

topmargin — In left inlet: The word topmargin, followed by an int greater than or equal to zero, sets the top margin, in pixels, for the **pictslider**. The margin reduces the area in which the knob moves; if a margin is zero, the knob can move all the way to the top of the slider.

topvalue In left inlet: The word topvalue, followed by an int, sets the values emitted by the **pictslider** object when the knob is moved as far as possible to the top. The message topvalue 100 will cause the control to send 100 out of its left outlet when the knob is moved all the way to the top.

track In left inlet: The word track, followed by a float, sets the tracking ratio for horizontal movements of the **pictslider** object's knob.

In right inlet: The word track, followed by a float, sets the tracking ratio for vertical movements of the **pictslider** object's knob.

verticaltracking In left inlet: The word verticaltracking, followed by a float, sets the vertical tracking ratio for movements of the **pictslider** object's knob. The default value is 1.0. Values greater than one cause the knob to move more quickly when dragged; values less than one cause it to move more slowly.

Inspector

The behavior of a **pictslider** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **pictslider** object displays the **pictslider** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **pictslider** Inspector lets you set the following attributes:

The *Margin* number boxes set the corresponding margin for the **pictslider**, in pixels. The margins reduce the area in which the knob moves. If a margin is zero, the knob can move all the way to the corresponding edge of the slider. If the left margin is five, for example, the knob can move no closer than five pixels to the left edge of the slider.

The *Value* number boxes set the values emitted by the control when the knob is moved as far as possible in the corresponding direction. For example, setting the right-hand number box to 100 will cause the control to send 100 out of its left outlet when the knob is moved all the way to the right. (The value is sent out the left outlet because the left outlet emits values for horizontal movements of the knob.) Values for intermediate positions of the knob are calculated by interpolating between the left and right or top and bottom values. Either one of each pair of numbers can be larger, so for example if the top value is -100 and the bottom is 50, the vertical value will decrease from 50 to -100 as the knob is moved from the bottom to the top.

If the *Move Horizontal* or *Move Vertical* checkboxes are checked, the knob can be moved in the corresponding direction by clicking and dragging it with the mouse. If you're creating a traditional slider that moves only horizontally or vertically, check the appropriate checkbox and leave the other unchecked.

Selecting the *Knob Jumps to Click Location* option lets you click anywhere within the **pictslider** object's bounding rectangle and have the knob jump to this location. If unchecked, you must click and drag the knob itself to move it.

The *Has Inactive Images* checkbox tells the **pictslider** object that your graphics files have additional images for the control's inactive state. Leave this box unchecked if the picture files used by the control do not have these images.

The *Tracking Ratio* values determine the responsiveness of the knob to mouse movements. The default value is 1.0. Values greater than one cause the knob to move more quickly when dragged; values less than one cause it to move more slowly.

There are four attributes listed in the Inspector that let you change the appearance of the slider's knob. You can choose a graphics file for the slider's knob by clicking on the Open button. The current file's name appears in the text box to the left of the button. You can also choose a file by typing its name in this box, or by dragging the file's icon from the Finder into this box.

Checking the *Scale Knob When Control Size Changes* option allows the knob's image to be stretched or compressed when you resize the **pictslider**, in proportion to the relative sizes of the object's bounding box and the background picture. If unchecked, the knob's image will be drawn at its original size. Since stretched images tend to look blocky and uneven, you will usually want to draw an image for your knob at the size that you want the knob to be. This knob-scaling attribute is useful for experimenting with the size and layout of the **pictslider** without having to redraw the knob's picture file.

Checking the *Clicked Image* option will use an alternate set of image frames in your picture file to give the knob a different appearance when the user clicks and drags it.

If you want to use image masks in your knob's graphics file to draw the knob, select the *Image Mask* option. Masks can be used to create knobs with a non-rectangular shape. If your knob picture has separate images for the clicked and/or inactive state, you must supply masks for those as well.

There are three attributes listed in the Inspector that let you change the appearance of the slider's background. You can choose a graphics file for the slider's background by clicking on the Open button. The current file's name appears in the text box to the left of the button. You can also choose a file by typing its name in this box, or by dragging the file's icon from the Finder into this box.

If *Size Control to Background Image* is checked, the pictctrl object's size is adjusted to match the size of the image chosen for the background. When this attribute is enabled, you cannot change the object's size in the usual manner by clicking and dragging its lower-right corner; its size is fixed. If unchecked, the image is stretched or shrunk to fill the size of the slider. Since stretched images tend to look blocky and uneven, you will usually want to draw an image for your slider at the size that you want the slider to be. Leaving this sizing attribute unchecked is useful for experimenting with the size and layout of the **pictslider** without having to redraw the slider's picture file.

Checking the *Invisible Background* box tells the **pictslider** object not to draw anything for the slider's background. The knob will appear to "float" over any underlying objects.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

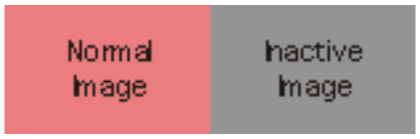
Arguments

None.

Picture File Format

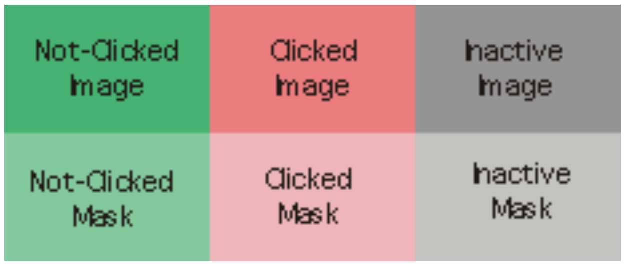
The **pictslider** object uses the two picture files: one for the background, and one for the knob that is moved over the background with the mouse.

Background picture files can be in PICT format, or if QuickTime Version 3.0 or later is installed, one of the other graphics file formats listed in the QuickTime appendix. Background picture files must have the following layout:



Only one image is required; if only one image is supplied, it will be used for drawing all states of the background. Additional images are placed to the right of the first image. You can add images for the inactive state of the control. The inactive image will be used after the control has received an active 0 message.

Knob files must be in PICT format with the following layout:



The picture is made up of a grid of one or more images. All images have the same width and height.

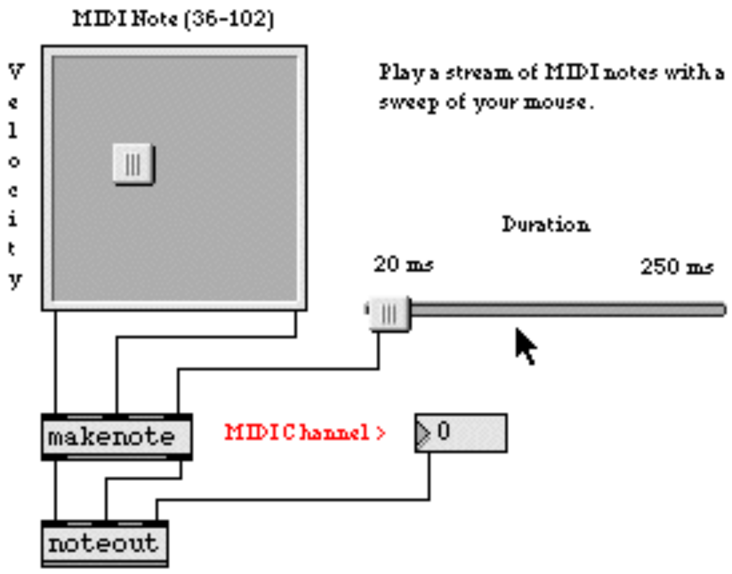
Only one image is required; if only one image is supplied, it will be used for drawing all states of the knob. Additional images are placed to the right of the first image. You can add images for either or both the "clicked" or inactive states of the control. The "clicked" image will be shown when the user is dragging the control's knob. The inactive image will be used after the control has received an active 0 message.

Image masks can be used to create knobs with non-rectangular outlines. These masks are directly below their corresponding images in the picture file. If you wish to use masks for any of the knob images, you must provide masks for all of them—each image will have a corresponding row of masks. Black pixels in the mask image create areas of the corresponding image that will be drawn, and white pixels create invisible areas.

Output

int Moving the slider's knob by clicking and dragging it with the mouse, or sending values to either of its inlets, causes its horizontal value to be emitted from the left outlet and its vertical value to be emitted from the right outlet. Incoming values are constrained to the ranges determined by the top/bottom and left/right values set in the inspector.

Examples



pictslider *lets you create both one- and two-dimensional UI elements*

See Also

dial	Output numbers by moving a dial onscreen
hslider	Output numbers by moving a slider onscreen
kslider	Output numbers from a keyboard onscreen
multislider	Multiple slider and scrolling display
pictctrl	Picture-based control
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
ubutton	Transparent button, sends a bang
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

pipe

Input

int — In left inlet: The number is delayed a certain number of milliseconds before it is sent out the left outlet. If there are middle inlets, the numbers in those inlets are also delayed and sent out the corresponding outlets.

int or float — In right inlets: Sets the time in milliseconds to delay numbers received in the other inlets.

bang — In left inlet: Retriggers the numbers currently stored in the **pipe** to be output again in the specified number of milliseconds (*in addition to* any numbers already being delayed).

float — In left and middle inlets: Converted to int, unless the inlet was initialized with a float argument.

list — In left inlet: Numbers are distributed to the **pipe** object's inlets to be delayed together. If there is a number for the right inlet, it sets the delay time for the other numbers.

clear — In left inlet: Halts all numbers currently being delayed by **pipe**.

clock — The word clock, followed by the name of an existing **setclock** object, sets **pipe** to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets **pipe** back to using Max's regular millisecond clock.

flush — In left inlet: Immediately sends out all numbers currently being delayed by **pipe**, and clears the **pipe** object's memory. Numbers are sent out each outlet in reverse order from that in which they were received in the corresponding inlet.

Arguments

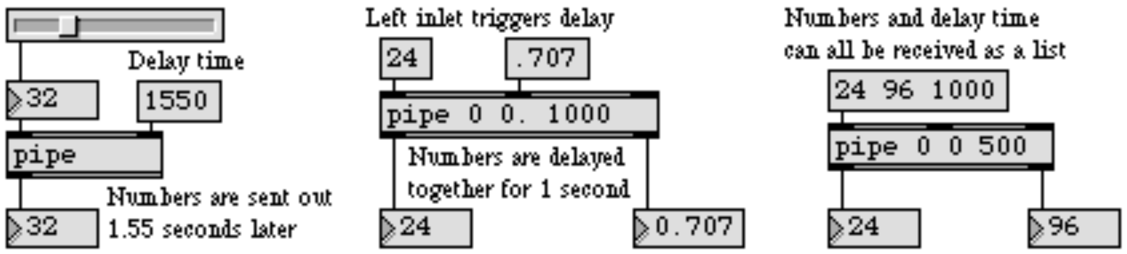
int — Optional. The last argument sets an initial value for the delay time, in milliseconds. If there is no argument, the delay time is 0. If there are two arguments, the first argument sets an initial value to be stored in **pipe**, and the second arguments sets the delay time. If more than two arguments are present, **pipe** creates additional inlets and outlets for delaying additional numbers in parallel to the leftmost one.

float — The last argument is converted to int. Other float arguments cause the corresponding outlet to send a float.

Output

int — When a number is received in the **pipe** object's left inlet, it is delayed by the time specified, then sent out the left outlet. If there are middle inlets, the numbers in those inlets are also delayed and sent out their corresponding outlet, in response to a number is received in the left inlet. Unlike **delay**, more than one number at a time can be delayed in a **pipe**. When a new delay time is received in the right inlet, it does not affect when the numbers already being delayed by **pipe** will come out.

Examples


Delay time
32
1550
pipe
Numbers are sent out 1.55 seconds later
32
Left inlet triggers delay
24
.707
pipe 0 0. 1000
Numbers are delayed together for 1 second
24
0.707
Numbers and delay time can all be received as a list
24 96 1000
pipe 0 0 500
24
96

One or more numbers can be delayed with **pipe**

See Also

delay	Delay a bang before passing it on
Tutorial 22	Delay lines

playbar

QuickTime movie play controller

Note: The **playbar** object requires that QuickTime be installed on your system. If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.

Input

bang	If the left outlet of a **playbar** object is connected to a **movie** or **imovie** object, bang links the two objects together so the **playbar** can control the QuickTime movie. After **playbar** and **movie** are linked, any messages sent to the **movie** object which change its location or playing status are reflected in the **playbar** object. (Linking will happen automatically when a patcher file containing connected **playbar** and **movie** objects is loaded. Thus, sending the bang to **playbar** is only necessary when you're building a patch.)

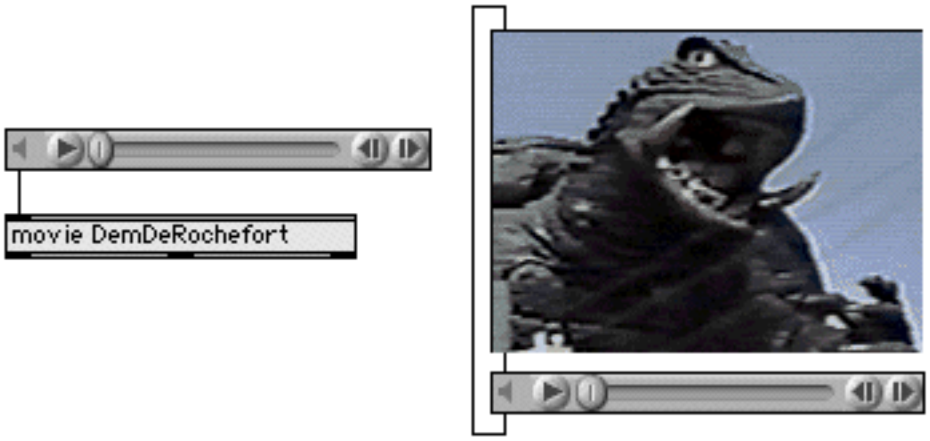
Arguments

None.

Output

(internal)	Out left outlet: Once the **playbar** and a **movie** object are linked, the **playbar** controls the QuickTime movie. **playbar** only supports being connected to one **movie** object at a time. The connection must be made with a patch cord; it cannot take place via a **send**-**receive** pair.
int	Out right outlet: Each command processed by **playbar** is sent by number out its right outlet. A directory of command numbers and their meaning can be found in the QuickTime Standard Movie Play Controller documentation. By properly interpreting these commands, you can potentially use **playbar** for other purposes besides movie control. However, the "thumb" in the controller has no range until an associated QuickTime movie with a non-zero duration is linked to the **playbar**.

Examples



Using **playbar** *with* **movie** *and* **imovie**

See Also

movie	Play a QuickTime movie in a window
imovie	Play a QuickTime movie in a patcher window

poltocar

Polar to Cartesian coordinate conversion

Input

float	In left inlet: The magnitude (amplitude) portion of a polar coordinate pair to be converted into a cartesian (real/imaginary) coordinate pair. In right inlet: The phase portion of a polar coordinate pair to be converted into a cartesian (real/imaginary) coordinate pair.
int	Converted to float.

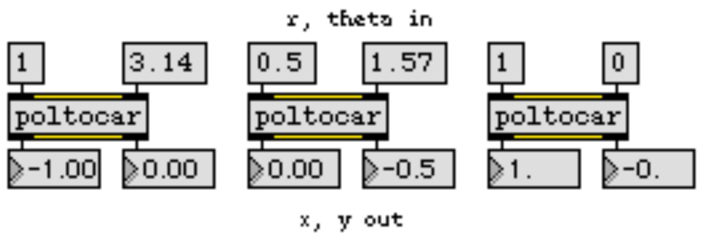
Arguments

None.

Output

float	Out left outlet: The real portion of a frequency domain coordinate pair. Out right outlet: The imaginary portion of a frequency domain coordinate pair.

Examples



Convert Polar to Cartesian coordinates

See Also

cos	Cosine function
cartopol	Cartesian to Polar coordinate conversion
lcd	Draw graphics in a Patcher window
sin	Sine function

poly

Allocate notes to different voices

Input

list	In left inlet: The first number is treated as a pitch, and the second number is treated as a velocity value, of a pitch-velocity pair. If the velocity is not 0, **poly** allocates that note-on to the first available voice number and sends it out. If the velocity is 0, **poly** frees the voice that is holding that pitch and sends out the note-off.
int	In left inlet: The number is treated as the pitch value of pitch-velocity pair and the note is sent out. In right inlet: The number is stored as the velocity to be paired with numbers received in the left inlet.
float	Converted to int.
stop	In left inlet: Immediately sends note-offs for all the notes currently being held by **poly**, freeing all voices.

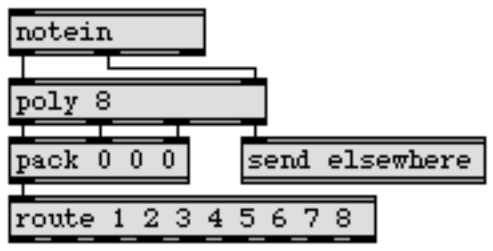
Arguments

int	Optional. The first argument sets the number of voices to which **poly** can allocate notes (thus limiting the number of notes **poly** can hold at one time). If there is no argument present, **poly** can hold 16 notes. If there is no second argument, or if the second argument is 0, **poly** sends any notes it cannot hold out the rightmost outlet. If there is a second argument not equal to 0, **poly** *steals voices*: when **poly** receives more notes than it has voices, it turns off the note it has held the longest and puts the new note in its place.
float	Converted to int.

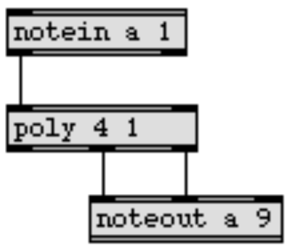
Output

int	Out left outlet: The output is the voice number of the note-on or note-off being sent out. Out 2nd outlet: The output is the pitch of the note-on or note-off. Out 3rd outlet: The number is the velocity of the note-on or note-off.
list	Out 4th outlet: The first number is the pitch, and the second number is the velocity, of any notes **poly** cannot hold. If there is a nonzero second argument, **poly** steals voices rather than send out overflow, so the fourth outlet is not created.

Examples



Send each voice to a different place



Limit the number of notes held at a time

See Also

borax	Report current information about note-ons and note-offs
flush	Provide note-offs for held notes
makenote	Generate a note-off message following each note-on

polyin

Output received MIDI poly pressure values

Input

(MIDI)	**polyin** receives its input from MIDI polyphonic key pressure messages received from a MIDI input device.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming polyphonic key pressure messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **polyin** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

a-z	Optional. Specifies the port from which to receive incoming MIDI messages. If there is no argument, **polyin** receives from all channels on all ports.
(MIDI name)	Optional. The name of a MIDI input device may be used as the first argument to specify the port.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to receive polyphonic key pressure messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

Output

int Out left outlet: The number is the pressure value of the incoming polyphonic key pressure message.

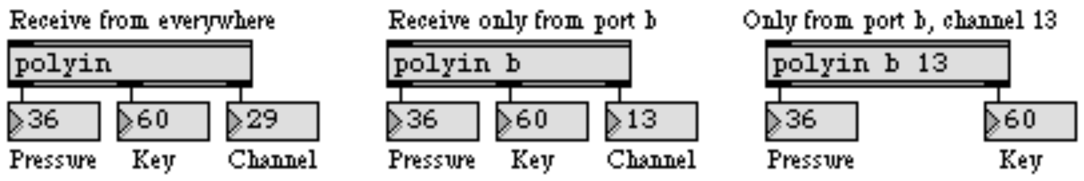
Out 2nd outlet: The number is the pitch value (key number) of the incoming message.

If a specific channel number is included in the argument, there are only two outlets. If there is *no* channel number specified by the argument, **polyin** will have a third outlet, on the right, which will output the channel number of the incoming note-on message.

polyin

Output received MIDI poly pressure values

Examples



Messages can be received from everywhere, a specific port, or a specific port and channel

See Also

midiin	Output received raw MIDI data
polyout	Transmit MIDI poly pressure messages
Tutorial 16	More MIDI ins and outs
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified

polyout

Transmit MIDI poly pressure messages

Input

int	In left inlet: The number is the pressure value of a MIDI polyphonic key pressure message transmitted on the specified channel and port. Numbers are limited between 0 and 127. In middle inlet: The number is stored as the key number, to be used with pressure values received in the left inlet. Numbers are limited between 0 and 127. In right inlet: The number is stored as the channel number on which to transmit the polyphonic key pressure messages.
float	Converted to int.
list	In left inlet: The first number is the pressure value, the second number is the key number, and the third number is the channel, of a transmitted MIDI polyphonic key pressure message.
enable	The message enable 0 disables the object, causing it not to transmit MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	In left inlet: The word port, followed by a letter a-z or the name of a MIDI output port or device, specifies the port used to transmit the polyphonic key pressure messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **polyout** object shows a pop-up menu for choosing a MIDI port or device.

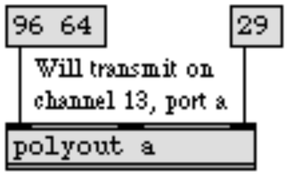
Arguments

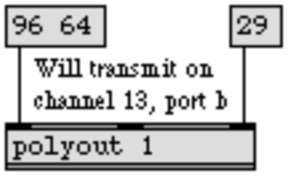
a-z	Optional. Specifies the port for transmitting MIDI polyphonic key pressure messages. Channel numbers greater than 16 received in the right inlet will be *wrapped around* to stay within the 1-16 range. If there is no argument, **polyout** initially transmits out port a, on MIDI channel 1.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to transmit polyphonic key pressure messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
(MIDI name)	Optional. The name of a MIDI output device may be used as the first argument to specify the port.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the **MIDI Setup** dialog.

Output

(MIDI)	There are no outlets. The output is a MIDI polyphonic key pressure message transmitted directly to the object's MIDI output port.

Examples





Letter argument transmits to only one port.
Otherwise, number specifies both port and channel

See Also

midiout	Transmit raw MIDI data
polyin	Output received MIDI poly pressure values
Tutorial 16	More MIDI ins and outs
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified

pow

Compute x to the power of y

pow raises the *base value* (set in the right inlet) to the power of the exponent (set in the left inlet).

Input

float or int	In left inlet: Sets the exponent.
	In right inlet: Sets the base value.

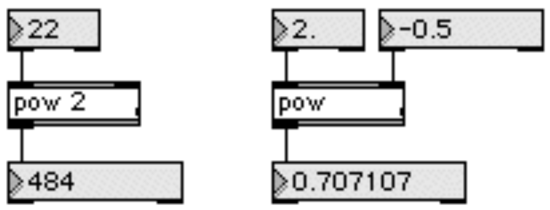
Arguments

float or int	Optional. Sets the base value. The default value is 0.

Output

float	The base value (from the right inlet) raised to the exponent (from the left inlet).

Examples



pow *will give you a square deal (and other numbers, too)*

See Also

expr	Evaluate a mathematical expression
>>	Shift all bits to the right
<<	Shift all bits to the left

prepend

Put one message at the beginning of another

Input

set	The word set, followed by any message, will replace the message stored in **prepend**, without triggering output.
anything else	The message stored in **prepend** is attached to the beginning of the message received in the inlet, and the combined message is sent out its outlet.

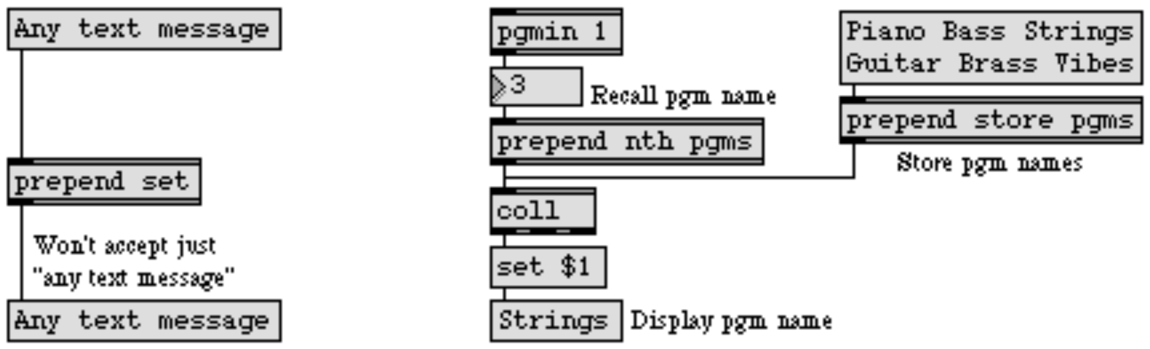
Arguments

anything	Obligatory. Sets the message to be prepended at the beginning of incoming messages. The first argument must be a symbol.

Output

anything	The message received in the inlet is combined with the message stored in **prepend**, and then sent out the outlet. The maximum allowed length of any constructed message is 256 items.

Examples



Symbols can be combined into meaningful messages with **prepend**

See Also

append	Append arguments at the end of a message
message	Send any message
route	Selectively pass the input out a specific outlet
Tutorial 25	Managing messages

preset

Store and recall the settings of other objects

Input

int	The number indicates a preset, and the settings stored in that preset are sent out to the connected objects, or to all objects in the window if no patch cords are connected to the **preset** object's outlet. The settings in a preset can also be sent out by clicking on the preset with the mouse.
float	Converted to int.
bang	Sends out the settings of the preset that was most recently recalled with an int or a mouse click.
clear	Erases the contents of the most recently sent preset. The word clear, followed by a number, erases the contents of that numbered preset.
clearall	Erases the contents of all presets.
list	Same as bang.
name	The word name, followed by a symbol, sets the ID Name for the preset. The ID Name allows the preset to have a unique ID so that files created for it will not read into other presets.
read	The word read, followed by no arguments or a number, displays an Open Document dialog box for choosing a file of preset data to read. If the preset has been given a Preset Name Code, only files of the type specified by the code will be displayed. The number argument specifies the preset number into which the file data should be read. If the number is 0 or -1, the data in the file will be read into the number of presets contained in the file starting with the first one. If the word read is followed by a symbol or a number and a symbol, no dialog box is displayed. Instead, the symbol is taken as a filename from which to read presets. The number functions as already described.
store	The word store, followed by a number, it stores the current setting of all user interface objects in the same window in the preset indicated by the number. If objects are connected to the **preset** object's left outlet with patch cords, only those connected objects will be affected.
	The presets (storage locations in the **preset** object) are numbered left-to-right, top-to-bottom. When settings are stored in a preset, a dot appears on it to indicate that it contains something. Settings can also be stored in a preset by holding down the Shift key and clicking on the preset with the mouse.
write	The word read, followed by no arguments or a number, displays a Save As dialog box for specifying a destination filename for writing the preset data. If the preset has been given a Preset Name Code, the file is given this code as its file type. The number argument specifies the preset number from which the preset data should be written. If the number is 0 or -1, all presets will be written. If the word write is

followed by a symbol or a number and a symbol, no dialog box is displayed. Instead, the symbol is taken as a filename to use for writing the data; the file will be placed in the current default folder The number functions as already described.

Inspector

The behavior of a **preset** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **preset** object displays the **preset** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **preset** Inspector lets you specify an *ID Name* to the **preset** object, to distinguish it from other **preset** objects. The first four characters of this name, if you enter one, are used as the Macintosh "file type" for files of presets saved by this object. When you send the read message to a **preset** object that has an ID Name, only the files whose types match the first four characters of this name are shown in the standard file dialog. This allows you to create a "document type" for preset files so the user won't open a preset file designed for another **preset** object. A **preset** object can also be set to save its contents as part of the patch that contains it by checking the *Save Presets with Patcher* check box.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

Output

int or float — Out left outlet: When a preset is recalled, either by a mouse click or by a number in the inlet, the settings stored in that preset are sent out the outlet to all connected objects, or, if no objects are connected, to all user interface objects in the window.

int — Out middle outlet: When a preset is recalled, the number of the preset is sent out.

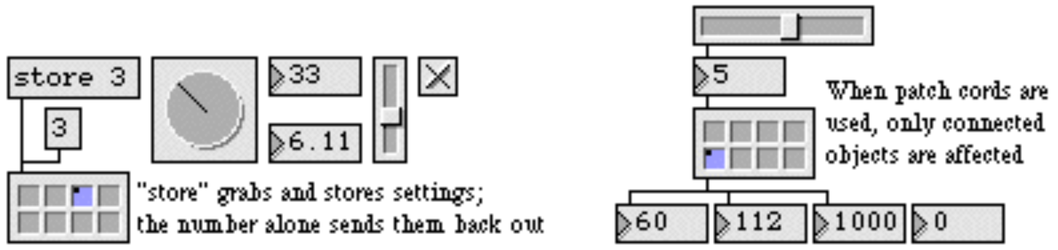
(internal) — Any user interface objects connected to the right outlet of **preset** will be excluded from the effects of that **preset**. (This is particularly useful when there are many objects you want to affect with **preset**, and only a few you want to exclude.)

Objects whose data is stored in a preset include: **dial**, **Ggate**, **Gswitch**, **hslider**, **led**, **number box** (both int and float), **slider**, **toggle**, and **uslider**. The contents of a **table** can also be stored and recalled by **preset**, but the **table** *must* be connected to the

preset object's outlet with a patch cord. The outlet of **preset** can also be connected to a **send** object, to communicate with objects connected to a **receive** object of the same name.

The number of visible presets can be adjusted by resizing the **preset** object's box. The maximum number of presets in a single **preset** object is 2048.

Examples


store 3
3
33
6.11
"store" grabs and stores settings;
the number alone sends them back out
5
When patch cords are used, only connected objects are affected
60
112
1000
0

Remember many past settings and recall them later

See Also

grab	Intercept the output of another object
Tutorial 37	Data structures
Data Structures	Ways of storing data in Max

print

Print any message in the Max window

Input

anything	Messages are not interpreted by the **print** object. They are simply printed verbatim in the Max window.
(mouse)	Double-clicking on any **print** object opens the Max window or brings it to the front.

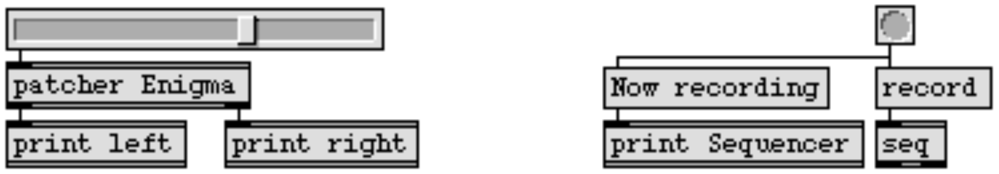
Arguments

anything	Optional. The argument is an identifier for the **print** object. Each message printed in the Max window is preceded by the name of the **print** object, and a colon (:). The name must not contain spaces or special characters, but can be either a number or a word. If there is no argument, the name of the **print** object is print. Using an argument to **print** can help distinguish the output of two or more **print** objects.

Output

anything	There are no outlets. The message received in the inlet is printed in the Max window.

Examples



Used for displaying output, or for notifying when an event takes place

See Also

Tutorial 1	Saying "Hello!"
Debugging	Techniques for debugging patches

prob

Make weighted random series of numbers

Input

list	The numbers make an entry in a probability matrix of transitions from one number to another (known as a *first-order Markov chain*). The list should consist of three numbers: a *current* value, a *next* value, and a probability that *current* will be followed by *next*. The first two numbers in the list identify a possible succession of output values: a possibility that the first number will be followed by the second. The third number sets the relative *likelihood* that the sequence of numbers will occur. Once the first number has been sent out, the next output is determined by the relative likelihood(s) assigned to each possible subsequent number.
bang	Makes a weighted random choice of a number to be sent out, based on the immediately previous output and on the specified likelihoods of subsequent numbers.
int	Sets (but does not send out) out the current number value. The subsequent output, in response to a bang message, will be determined by the stored matrix of probable transitions from that number.
reset	The word reset, followed by a number, tells **prob** what number to revert to in the event that it gets "stuck" on a number that has no possible next number.
dump	Prints out a complete list of the stored transition probabilities (Markov chain) in the Max window.
embed	The word embed, followed by a nonzero number, causes the contents of **prob** to be saved as part of the patch that contains it. The message embed 0 causes **prob** to forget its contents when the patch is closed.
clear	Erases the contents of **prob**.

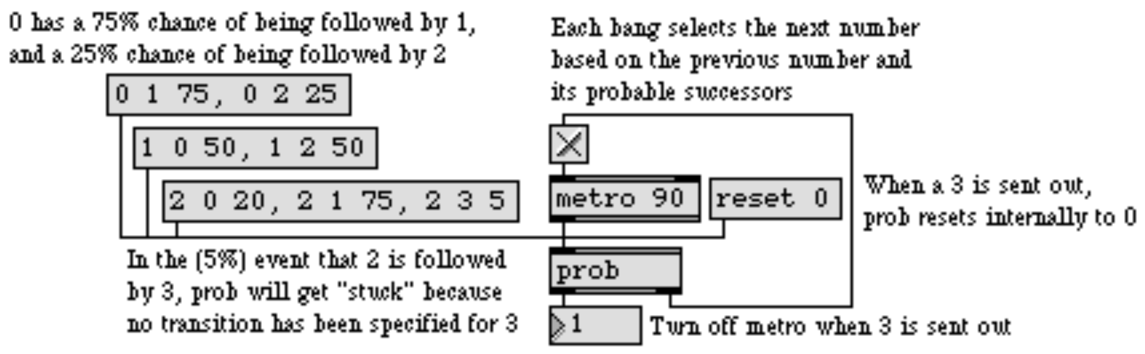
Arguments

None.

Output

int	Out left outlet: When bang is received in the inlet, **prob** sends out a number, which it chooses based on its knowledge of the last number chosen and the relative likelihood assigned to each possible subsequent number.
bang	Out right outlet: If the current number (the last number chosen) has no possible transitions listed in the transition probability matrix, bang is sent out (and nothing is sent out the left outlet) in response to a bang in the inlet.

Examples


0 has a 75% chance of being followed by 1, and a 25% chance of being followed by 2
0 1 75, 0 2 25
1 0 50, 1 2 50
2 0 20, 2 1 75, 2 3 5
In the (5%) event that 2 is followed by 3, prob will get "stuck" because no transition has been specified for 3
Each bang selects the next number based on the previous number and its probable successors
metro 90
reset 0
When a 3 is sent out, prob resets internally to 0
prob
1
Turn off metro when 3 is sent out

Likelihood of a certain output depends on the previous output

See Also

anal	Make a histogram of number pairs received
histo	Make a histogram of the numbers received
mean	Find the running average of a stream of numbers

pv

Share variables specific to a patch and its subpatches

pv operates identically to the **value** object, with two exceptions. First, **pv** objects that share the same name only share the same value if they are in the same patcher, or one of its subpatches. Second, the **pv** object cannot be the receiver of a message sent remotely by a **message** box (the first symbol after a semicolon). So, **pv** means *private value*—a value that is shared between objects, but only within a single patcher.

Input

any message	The message is stored, to be shared by all other **pv** objects of the same name that are inside the object's patcher or its subpatches (or, if in a subpatch, its parent patch). A message received in any other such **pv** object will change the stored message.
bang	Sends out the stored message.

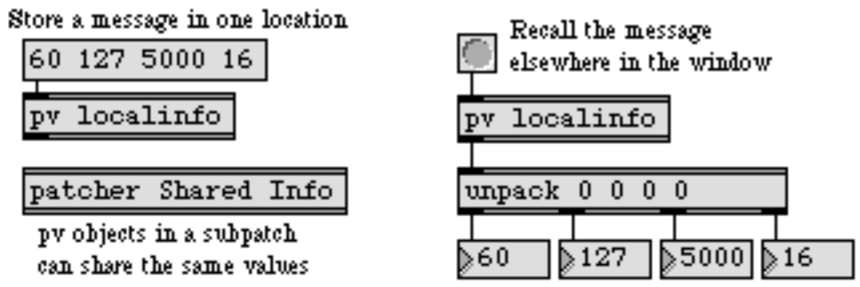
Arguments

any symbol	Obligatory. The first argument provides an identifying name. All **pv** objects with that name within the patcher will share the same value.
any message	Optional. Any message typed in after the first argument initializes the stored contents of the **pv** object. Note that when two or more **pv** objects in a patcher file that share the same name are initialized to different values, the one which is initialized last determines the value. Since the order in which **pv** objects will be initialized cannot be precisely determined, the best practice is to initialize only one of the related **pv** objects.

Output

any message	When bang is received in the inlet, the stored message is sent out.

Examples


Store a message in one location
60 127 5000 16
pv localinfo
patcher Shared Info
pv objects in a subpatch can share the same values
Recall the message elsewhere in the window
pv localinfo
unpack 0 0 0 0
60
127
5000
16

See Also

float	Store a decimal number
int	Store an integer value
pvar	Connect to a named object in a patcher
receive	Receive messages without patch cords
send	Send messages without patch cords
value	Share a stored message with other objects

pvar

Connect to a named object in a patcher

The **pvar** object lets you build user interfaces in one part of your patcher that are associated with the "process" part in another part of the patcher. Unlike the **send** and **receive** objects, **pvar** does not work globally; the **pvar** object and its associated object must be in the same patcher. You set an object's name by selecting the object and choosing Name Object from the Object menu. The name cannot be a number, although it can contain numbers.

Input

any message	The message is sent to the named object currently associated with **pvar**.
setname	The word setname, followed by a symbol, specifies the name of the object to which **pvar** will be associated with. The named object must be in the same patcher as the **pvar** object.

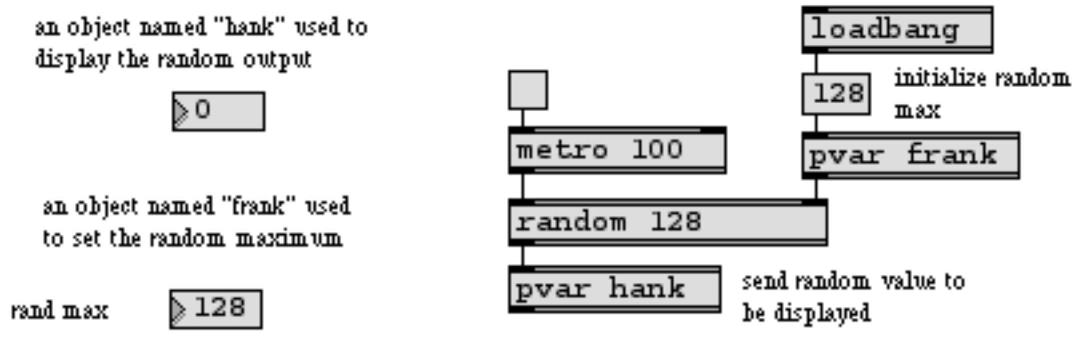
Arguments

symbol	Optional. The first argument specifies the name of the object to which **pvar** will be associated with. If no name is supplied, the setname message can be used to connect later.
int	Optional. The second argument specifies the number of outlets **pvar** will have. **pvar** connects to as many outlets as its associated object has, unless it is more than the number you specify as an argument. The default number of outlets is 1.

Output

any message	The outlets of **pvar** correspond to the outlets of its associated named object. When the named object sends anything out one of its outlets, the output also comes out of the corresponding outlets of the **pvar** object.

Examples



an object named "hank" used to display the random output
0
an object named "frank" used to set the random maximum
rand max
128
loadbang
128
initialize random max
metro 100
pvar frank
random 128
pvar hank
send random value to be displayed

pvar *can be used to build a user interface without any messy patch cords*

See Also

receive	Receive messages without patch cords
send	Send messages without patch cords
thispatcher	Send messages to a patcher
value	Share a stored message with other objects

radiogroup

Radio button/check box user interface object

The radiogroup object has two modes of operation: *radio button* and *check box*. In radio button mode, the **radiogroup** object provides a user-definable number of buttons in a group, only one of which may be selected at a time. In check box mode, the indicators in the **radiogroup** object function as a set of on/off indicators. Check box mode also supports a way to have the checkboxes act as indicators for the bit pattern of a binary representation of an integer (see the flagmode message below).

Note: **radiogroup** can be re-sized horizontally so it will extend under comment boxes placed to the right of the buttons or boxes. this way, clicking on the text to the right of the button will also set the button selection or box state.

Input

(mouse) In *radio button* mode, clicking on a radio button will set the radio button selection and output the corresponding button number (numbering starts from 0).

In *check box* mode, clicking on a check box will change its state (from 1 to 0 or from 0 to 1) and output a list of zeros and ones corresponding to the on/off state of the boxes. if the entire group of buttons/boxes is inactive (greyed out) it will not respond to clicks. if an individual item is disabled (greyed out) it will not respond to clicks, although active items in the group will still respond to clicks as usual. The Flag Mode variation on the check box mode has check boxes that correspond to bit positions for a binary value (i.e. the first checkbox corresponds to the 1s, the second to 2s, the third to 4s, etc.) Clicking on a check box will select or deselect the check box and output the integer value which corresponds to the bit pattern.

bang In radio button mode: A bang outputs the currently selected radio button number.

In check box mode: A bang outputs a list of zeros and ones representing the on/off state of the check boxes.

In flag mode: A bang send the integer that corresponds to the bit pattern of the currently checked boxes (i.e., if boxes one, two, and three are checked, a bang will output a value of 7)out the **radiogroup** object's output.

int In radio button mode: An integer sets the radio button selection and outputs the input value. Numbering starts with 0, and a negative number indicates that no buttons will be selected.

In flag mode: An integer value received in the **radiogroup** object's inlet will set the buttons or checkboxes to reflect the bit pattern of the integer value (i.e., a value of 19 will select boxes one, two, and five, corresponding to the binary value 10011) and send the integer value out the **radiogroup** object's output.

float In radio button and check box modes: Converted to int.

list In check box mode: list of zeros and ones sets the check box states and causes output of the input list. If you have specified check box mode and have the flag mode set using the flagmode 1 message, a list of zeros and ones sets the check box states and causes output of the input list.

disableitem
In radio button and check box modes: disable the items whose numbers are indicated (they will be drawn in grey and will not respond to clicks, although they will still respond to set messages, ints or lists).

enableitem
In radio button and check box modes: The word enableitem, followed by followed by a number or list of numbers, will enable the items whose numbers are indicated if they have been disabled with the disableitem message.

flagmode
In check box mode: The word flagmode, followed by a nonzero value, sets the *flag mode* of operation for the **radiogroup** object. In this mode, each check box corresponds to one bit in an integer value (i.e., the first radio button or checkbox corresponds to the ones bit, the second button or checkbox to the twos bit, the third button or checkbox to the fours bit, etc.). The message flagmode 0 disables this mode (default).

itemtype
In radio button and check box modes: The word itemtype, followed by a zero or one, selects the mode of the **radiogroup** object. The message itemtype 0 selects radio button mode, and itemtype 1 selects check box mode.

inactive
In radio button and check box modes: The word inactive, followed by a zero or one, toggles the active or inactive state of the entire group of radio buttons or check boxes. inactive 0 (default) means that the boxes are not inactive, and will respond to mouse clicks. The message inactive 1 will gray out the radio buttons or check box displays, and they will not respond to mouse clicks (although their state can still be set using set messages, ints or lists).

offset
In radio button and check box modes: The word offset, followed by a number, changes the pixel offset between the tops of the buttons/boxes. the minimum offset is 14 pixels, the default offset is 16 pixels.

set
In radio button mode: The word set, followed by a number, sets the currently selected radio button without triggering any output.

In check box mode: The word set, followed by a list of zeros and ones, sets the check box states without triggering any output.

If you are using check box mode and are also using Flag Mode, a number will set the state of the first 32 checkboxes in a pattern which corresponds to the bit pattern of the number without triggering output (see the flagmode section for more information).

size
In radio button and check box modes: The word size, followed by a number, changes the number of buttons or boxes. The default is 2, and the maximum is 64.

Note: If you care using the radiogroup object in check box mode and have enabled Flag Mode, you will only be able to set 32 checkboxes.

Inspector

The behavior of a **radiogroup** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **radiogroup** object displays the **radiogroup** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **radiogroup** Inspector lets you specify the *Number of Buttons* (default 2) and their *Offset* (default 16 pixels). The *Button Type* option lets you choose between radio buttons (the default). If you choose the *Check Boxes* option, you can also specify the *Flag Mode* option (default is unchecked).

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

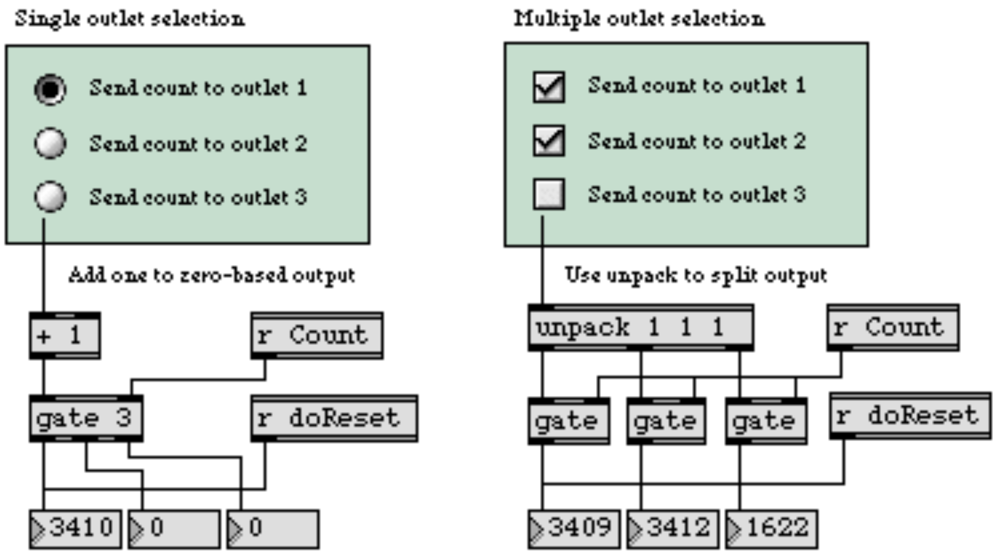
Output

int In radio button mode: Clicking on a radio button outputs an int corresponding to the radio button selected. Numbering begins with 0.

In flag mode: Clicking on a check box outputs an int corresponding to the bit pattern represented by the checked boxes (i.e., if boxes one, two, and three are checked, a bang will output a value of 7).

list In check box mode: A bang will output a list of zeros and one which indicate the on/off state of the group of check boxes.

Examples



Radio buttons allow a single selection, and multiple selection check boxes can control several gates

See Also

button	Flash on any message, send a bang
matrixcrtrl	Matrix-style switch control
pictctrl	Picture-based control
toggle	Switch between on and off (1 and 0)
ubutton	Transparent button, sends a bang

random

Generate a random number

Input

bang	In left inlet: Sends out a randomly generated number between 0 and one less than its maximum limit.
int	In right inlet: The number is stored as the maximum limit for the random output. The output will always be between 0 and one less than this maximum limit.
seed	In left inlet: The word seed, followed by a number, provides a "seed" value for the random generator, which causes a specific (reproducible) sequence of pseudo-random numbers to occur. The number 0 uses the time elapsed since system star-tup (an unpredictable value) as the seed, ensuring an unpredictable sequence of numbers. This unpredictable seed is used by default when the **random** object is created.

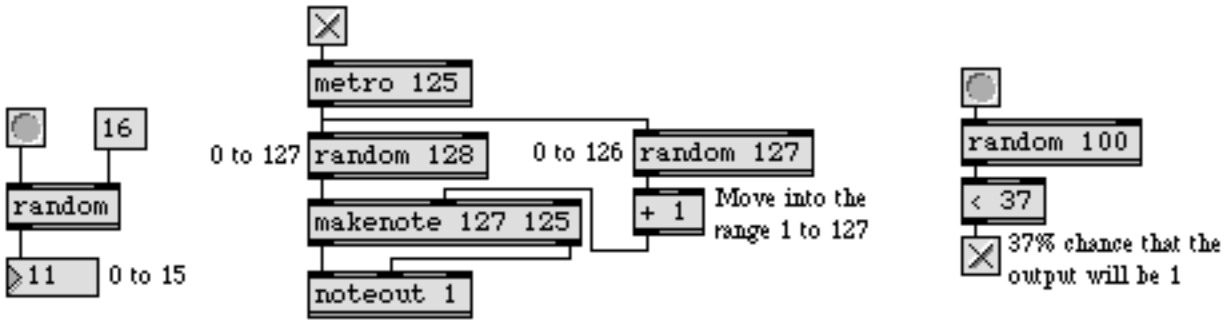
Arguments

int	Optional. Sets an initial limit to the random output. The output will always be between 0 and one less than this maximum limit. If there is no argument, the limit is initially set to 1, which causes **random** to output 0 whenever it receives a bang.
int	Optional. A second argument is used to set a "seed" value for the random generator. If no argument is specified, the time value will be used to initialize the seed.

Output

int	When a bang is received in the left inlet, **random** generates a random number between 0 and one less than its maximum limit.

Examples



Generate random events, or make decisions based on probability

random

Generate a random number

See Also

decide	Choose randomly between on and off (1 and 0)
drunk	Output random numbers in a moving range
urn	Generate random numbers without duplicates
Tutorial 22	Delay lines

receive / r

Receive messages without patch cords

Input

anything — Input is received from **send** or **forward** objects that have the same name, even if the sending object is in another loaded patch. The order in which multiple **receive** objects with the same name will sendout the message received is undefined, so the order in which their output will be sent out is unpredictable.

Messages can also be sent remotely to a **receive** object from an **int** or **float** object (with the word send followed by the name of the **receive** object), from a **grab** object (with a symbol argument), or from a **message** box (with a semicolon followed by the name of the **receive** object.

(mouse) — Double-clicking on a **receive** object looks for and opens a loaded patcher window containing a **send** object with the same name. Repeatedly double-clicking on the **receive** object looks for and opens more such windows.

set — If there is no typed-in argument, **receive** has one inlet. The word set, followed by a symbol, provides a name for **receive**, as if that name had been typed in as an argument.

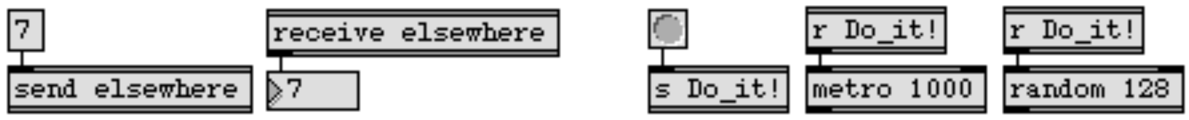
Arguments

any symbol — Optional. Gives a name to **receive**. If there is no argument, **receive** has one inlet, and a name must be provided by a set message before anything can be received.

Output

anything — Any message received in the inlet of any **send** or **forward** object with the same name, or sent explicitly from an **int**, **float**, **grab**, or **message** box, is passed out the outlet of **receive**, even if the sending object is in a different loaded patch.

Examples



Virtual connections exist between all **send** *and* **receive** *objects that share the same name*

See Also

float	Store a decimal number
forward	Send remote messages to a variety of objects
int	Store an integer value
message	Send any message
pvar	Connect to a named object in a patcher
route	Selectively pass the input out a specific outlet
send	Send messages without patch cords
value	Share a stored message with other objects
Tutorial 24	**send** and **receive**

rect

Draw solid rectangle in a graphic window

Input

bang — In left inlet: Draws the rectangle using the current screen coordinates, drawing mode, and color.

int — In left inlet: Sets the left screen coordinate of the rectangle—relative to the upper left corner of the graphics window—and draws the shape.

In 2nd inlet: Sets the top screen coordinate of the rectangle.

In 3rd inlet: Sets the right screen coordinate of the rectangle.

In 4th inlet: Sets the bottom screen coordinate of the rectangle.

In 5th inlet: Sets the drawing mode of the rectangle. The following are drawing mode constants; not all modes will be available on all operating systems.

Copy	0	blend	32
Or	1	addPin	33
Xor	2	addOver	34
Bic	3	subPin	35
NotCopy	4	transparent	36
NotOr	5	adMax	37
NotXor	6	subOver	38
NotBic	7	adMin	39

In 6th (right) inlet: Sets the palette index (color) of the rectangle according to the graphics window's current palette. This setting has no effect when the monitor is in black and white mode.

frgb — In left inlet: The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the color of the rectangle the next time it is drawn.

priority — In left inlet: The word priority, followed by a number greater than 0, sets a **rect** object's sprite priority in its graphics window. Objects with lower priority will draw behind those with a higher priority.

Arguments

any symbol — Obligatory. The first argument to **rect** must be the name of a graphics window into which the rectangle will be drawn. The window need not necessarily exist at the time the **rect** object is created, but the rectangle will not be drawn unless the name matches that of a visible window.

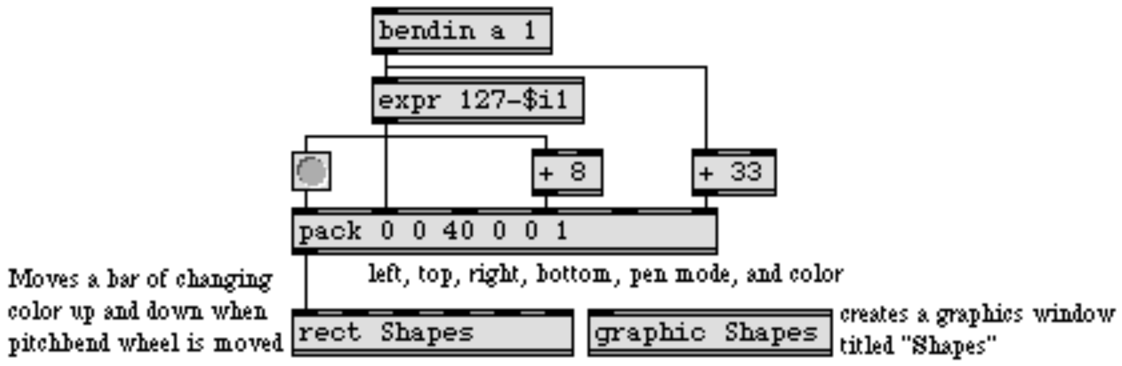
int — Optional. Sets the initial sprite priority of the **rect**. If no priority is specified, the default is 3.

rect

Output

(visual) When the **rect** object's associated graphics window is visible, and a bang message or number is received in its left inlet, a shape is drawn in the window, and the object's previously drawn rectangle (if any) is erased.

Examples


bendin a 1
expr 127-$i1
+ 8
+ 33
pack 0 0 40 0 0 1
left, top, right, bottom, pen mode, and color
Moves a bar of changing color up and down when pitchbend wheel is moved
rect Shapes
graphic Shapes
creates a graphics window titled "Shapes"

A rectangle can move in time with MIDI data or any other source of changing numbers

See Also

frame	Draw framed rectangle in a graphic window
graphic	Window for drawing sprite-based graphics
lcd	Draw graphics in a patcher window
oval	Draw solid oval in a graphic window
ring	Draw framed oval in a graphic window
Graphics	Overview of Max graphics windows and objects

relativepath

Convert an absolute to a relative path

Input

symbol	An absolute pathname of a folder or file as a symbol. An absolute pathname looks like this: 'MyDisk:/Max Folder/extras/filename'

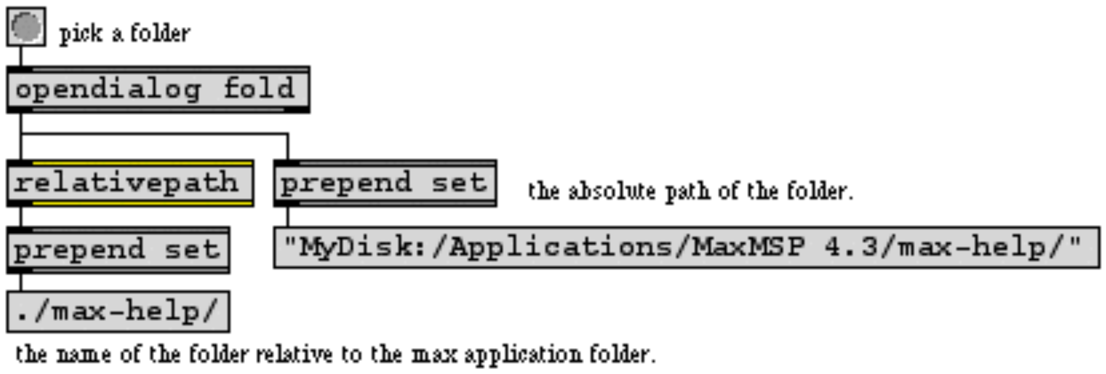
Arguments

None.

Output

symbol	The pathname of the folder or file relative to the Max application folder as a symbol. If the input pathname is within the Max application folder, the path is changed to start with a dot-slash (./) followed by the folder names of the path. Otherwise, the input is repeated to the output.

Examples


pick a folder
opendialog fold
relativepath
prepend set
prepend set
the absolute path of the folder.
"MyDisk:/Applications/MaxMSP 4.3/max-help/"
./max-help/
the name of the folder relative to the max application folder.

See Also

absolutepath	Convert a file name to an absolute path
conformpath	Convert paths of one pathtype and/or pathstyle to another
opendialog	Open a dialog to ask for a file or folder
strippath	Get a filename from an absolute pathname

ring

Draw framed oval in a graphic window

Input

bang — In left inlet: Draws a framed oval using the current screen coordinates, drawing mode, and color.

int — In left inlet: Sets the left screen coordinate of the oval—relative to the upper left corner of the graphics window—and draws the shape.

In 2nd inlet: Sets the top screen coordinate of the oval.

In 3rd inlet: Sets the right screen coordinate of the oval.

In 4th inlet: Sets the bottom screen coordinate of the oval.

In 5th inlet: Sets the drawing mode of the oval. The following are drawing mode constants; not all modes will be available on all operating systems.

Copy	0	blend	32
Or	1	addPin	33
Xor	2	addOver	34
Bic	3	subPin	35
NotCopy	4	transparent	36
NotOr	5	adMax	37
NotXor	6	subOver	38
NotBic	7	adMin	39

In 6th (right) inlet: Sets the palette index (color) of the oval according to the graphics window's current palette. This setting has no effect when the monitor is in black and white mode.

frgb — In left inlet: The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the color of the ring the next time it is drawn.

priority — In left inlet: The word priority, followed by a number greater than 0, sets a **ring** object's sprite priority in its graphics window. Objects with lower priority will draw behind those with a higher priority.

Arguments

any symbol — Obligatory. The first argument to **ring** must be the name of a graphics window into which the oval will be drawn. The window need not necessarily exist at the time the **ring** object is created, but the oval will not be drawn unless the name matches that of a visible window.

int — Optional. Sets the initial sprite priority of the **ring**. If no priority is specified, the default is 3.

ring

Draw framed oval in a graphic window

Output

(visual)	When the **ring** object's associated graphics window is visible, and a bang message or number is received in its left inlet, a shape is drawn in the window, and the object's previously drawn oval (if any) is erased.

Examples

See examples under **oval** or **rect**. **ring** can be directly substituted for **oval**, **rect**, or **frame**.

See Also

frame	Draw framed rectangle in a graphic window
graphic	Window for drawing sprite-based graphics
lcd	Draw graphics in a patcher window
oval	Draw solid oval in a graphic window
rect	Draw solid rectangle in a graphic window
Graphics	Overview of Max graphics windows and objects

route

Selectively pass the input out a specific outlet

Input

anything — If the first item of the message is the same as one of the arguments of **route**, the rest of the message is sent out the outlet that corresponds to that argument. If the first item does not match any of the arguments, the entire message is passed out the rightmost outlet.

Arguments

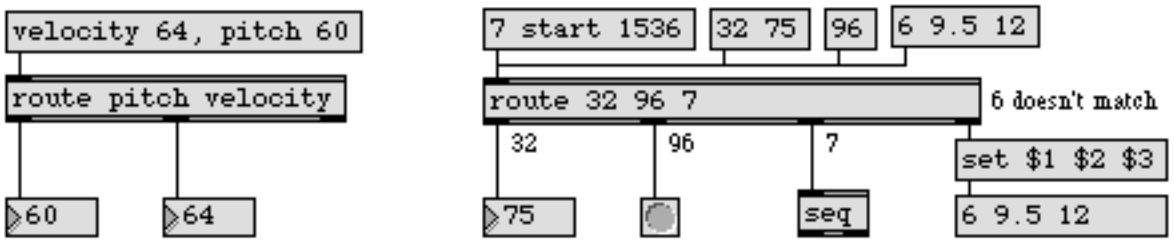
anything — Optional. Arguments can be a mix of ints, floats, or symbols. The number of arguments determines the number of outlets, *in addition to* the rightmost outlet. Each argument assigns a name or a number to an outlet. If there is no argument, there is one other outlet, which is assigned the number 0.

Output

anything — The first item of any message received in the inlet is compared with the arguments. If it matches one of the arguments, the rest of the message is sent out the specified outlet. Otherwise, the entire message is passed out the rightmost outlet.

bang — If the first item of a message matches one of the arguments, but the message has no additional items, bang is sent out the specified outlet.

Examples



Arguments assign names or numbers to the outlets, and route the input to the correct outlet

See Also

bucket	Pass a number from outlet to outlet, out each one in turn
forward	Send remote messages to a variety of objects
gate	Pass the input out a specific outlet
pack	Combine numbers and symbols into a list
receive	Receive messages without patch cords
select	Select certain inputs, pass the rest on
send	Send messages without patch cords
sprintf	Format a message of words and numbers
zl	Multi-purpose list processor
Tutorial 17	Gates and switches

rslider

Display or change a range of numbers

Input

int In left inlet: The number sets the minimum limit of a range displayed as a colored region on the **rslider**, and causes the minimum and maximum values of that range to be sent out. A number that exceeds the limits of the **rslider** itself will be limited to stay within the **rslider**.

In right inlet: The number is stored as the maximum limit of the range displayed in color on the **rslider**. A number that exceeds the limits of the **rslider** itself will be limited to stay within the **rslider**.

The minimum and maximum values can also be set (and sent out) by dragging with the mouse across a range in the **rslider**.

list In left inlet: The first two numbers in the list are used to set the minimum and maximum values of the displayed range, and are sent out.

bang In left inlet: Sends out the minimum and maximum values of the currently displayed range.

color The word color, followed by a number from 0 to 15, specifies a color for the range being displayed in the **rslider**—one of the object colors which are also available via the Color command in the Object menu.

float Converted to int.

mult In left inlet: The word mult followed by a number, specifies a multiplier value. The **rslider** object's value will be multiplied by this number before it is sent out the outlet. The default value is 1.

set In left inlet: The word set, followed by two numbers, sets the minimum and maximum values of the currently displayed range, without sending them out the outlets.

size In left inlet: The word size, followed by a positive number, determines the total range of the **rslider**. The **rslider** will range from 0 to one less than the specified size. A size message smaller than 1 will be automatically set to 2. By default, the size of an **rslider** is 128.

Inspector

The behavior of an **rslider** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **rslider** object displays the **rslider** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **rslider** Inspector lets you enter a *Maximum* value. Numbers received in the inlet are automatically limited between 0 and the number 1 less than the specified maximum value. The default range value is 128. The **rslider** Inspector also lets you specify a *Multiplier*. The **rslider** object's value will be multiplied by this number before it is sent out the outlet. The default multiplier value is 1.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

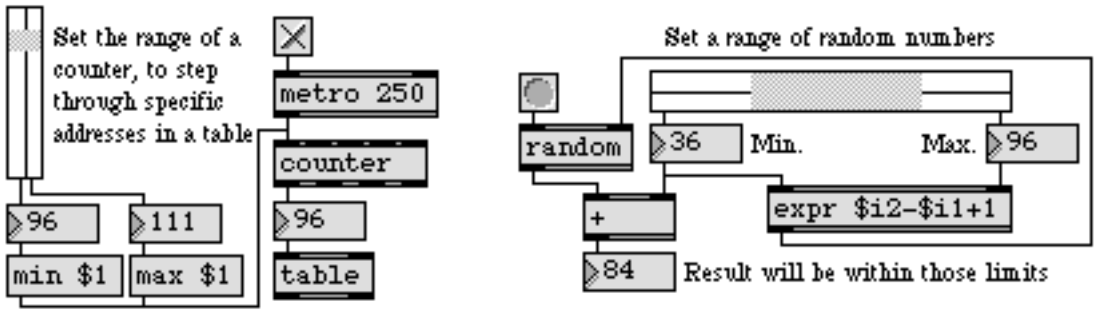
Arguments

None.

Output

int The maximum value of the displayed range is sent out the right outlet, and the minimum value is sent out the left outlet. Output is triggered by a new minimum value (or a bang) received in the left inlet, or by clicking or dragging the mouse in the **rslider**.

Examples


Set the range of a counter, to step through specific addresses in a table
metro 250
counter
96
111
96
min $1
max $1
table
Set a range of random numbers
random
36
Min.
Max.
96
+
expr $i2-$i1+1
84
Result will be within those limits

Output minimum and maximum values, to set the range of another object

See Also

hslider	Output numbers by moving a slider onscreen
multislider	Multiple slider and scrolling display
pictctrl	Picture-based control
pictslider	Picture-based slider
slider	Output numbers by moving a slider onscreen
split	Look for a range of numbers
uslider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials

rtin

Input

(MIDI)	**rtin** receives MIDI real time messages received from a MIDI input device.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming MIDI messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on an **rtin** object shows a pop-up menu for choosing a MIDI port or device.

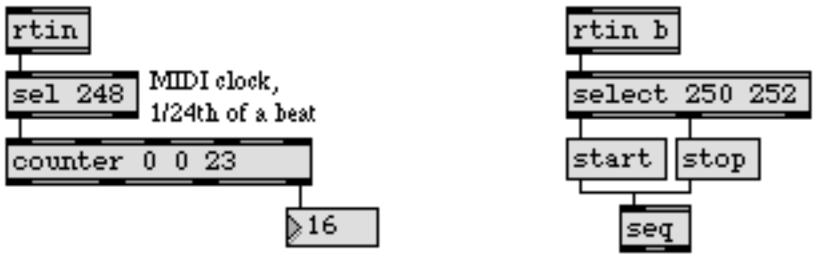
Arguments

a-z	Optional. Specifies the port from which to receive incoming MIDI real time messages. If there is no argument, **rtin** receives from port a (or the first input port listed in the MIDI Setup dialog.)

Output

int	MIDI real time messages (MIDI clock, start, stop, and continue) received from the specified port are sent out the outlet.

Examples



MIDI real time messages can be used to synchronize Max with external events

See Also

clocker	Report elapsed time, at regular intervals
metro	Output a bang message at regular intervals
midiin	Output received raw MIDI data
seq	Sequencer for recording and playing MIDI
MIDI	MIDI overview and specification
Using MIDI	Using Max with MIDI
Tutorial 16	More MIDI ins and outs

savedialog

Open a dialog to ask for a filename for saving

Input

bang	Causes a standard Save As dialog box to appear, allowing the user to type in a filename and choose a folder location. The resulting location and filename are output as a symbol.
set	The word set, followed by a four-letter symbol (e.g., TEXT, MAXB) which specifies a file type, sets the **savedialog** object to display the desired file type without opening the dialog box. The chosen file type is sent out the middle outlet when the user chooses Save in the dialog box.
anything	One or more four-letter type codes sets the list of types displayed in the dialog box. Example type codes for files are TEXT for text files, maxb for Max binary format patcher files, and AIFF for AIFF format audio files. The symbol fold specifies that the dialog box should let the user choose only folders.

Arguments

anything	Optional. Sets one or more file types that will be displayed as choices for the user. The symbol fold specifies that the dialog box should let the user choose only folders.

Output

symbol Out left outlet: The absolute pathname of the file as a symbol. The output pathnames contain slash separators.

Absolute pathnames look like this:

“C:/Max Folder/extras/mystuff/mypatch.pat”

The **conformpath** object can be used to convert paths of one pathtype and/or pathstyle to another.

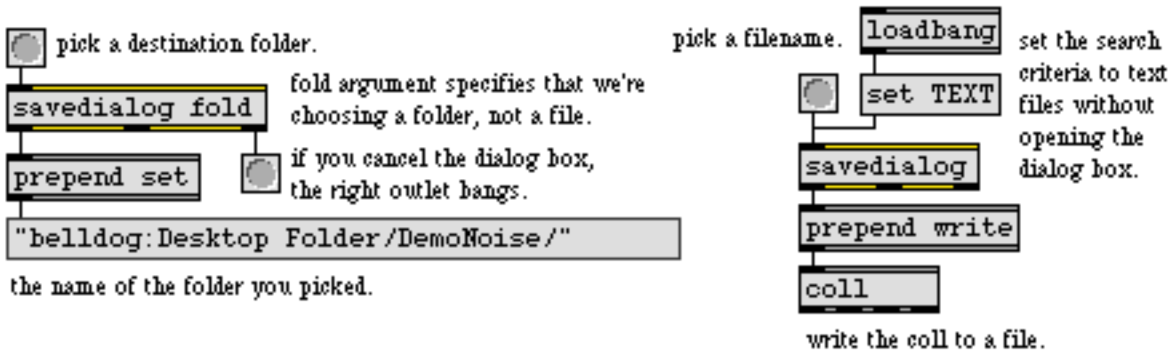
symbol Out middle outlet: The four-letter symbol which specifies the filetype currently selected.

bang Out right outlet: If the user chooses Cancel in the dialog box, a bang is sent out.

savedialog

Open a dialog to ask for a filename for saving

Examples


pick a destination folder.
savedialog fold
fold argument specifies that we're choosing a folder, not a file.
prepend set
if you cancel the dialog box, the right outlet bangs.
"belldog:Desktop Folder/DemoNoise/"
the name of the folder you picked.
pick a filename.
loadbang
set TEXT
set the search criteria to text files without opening the dialog box.
savedialog
prepend write
coll
write the coll to a file.

Select a folder or a specific file type for file saving

See Also

conformpath	Convert paths of one pathtype and/or pathstyle to another
dialog	Open a dialog box for text entry
filedate	Report the modification date of a file
filein	Read in a file of binary data
filepath	Report information about the current search path
opendialog	Open a dialog to ask for a file or folder

scale

Maps input range of values to output range

This object is not available in Max 4.1 and earlier.

Input

int	Converted to float.
float	In left inlet: The incoming value is scaled according to the mapping provided by the arguments, or values received in the other inlets. In second inlet: Sets the low input value. In third inlet: Sets the high input value. In fourth inlet: Sets the low output value. In fifth inlet: Sets the high output value. In right inlet: Sets the base value for exponential scaling. The minimum value is 1.0 which implies linear scaling. An appropriate value is 1.06.
bang	In left inlet: Performs the scaling operation on the previous input value. If the scaling ranges have changed since the previous input in the left inlet, the new ranges will be used for the scaling.

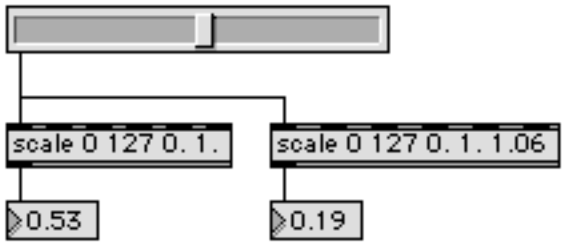
Arguments

int or float	Optional. The first argument is the minimum input value, the second argument is the maximum input value. The third and fourth arguments are the minimum and maximum output values, respectively. An optional fifth argument specifies the nature of the scaling curve. The greater the value is from 1 the more steeply exponential the curve is. A value of 1 (or less) indicates linear scaling.Only positive floating-point numbers greater than 1 are appropriate for the fifth argument. These five values can be changed via the object's five inlets. If only four arguments are provided and all four are of type int, scale will output integer values.

Output

int	If only four arguments are provided and all four are of type int, scale will output scaled values as integers. Otherwise output is floating-point.
float	When scale receives a value in its leftmost inlet, that value is scaled to the indicated output range of values.

Examples



An example of how to scale an integer slider into a useful range of floating-point values

See Also

zmap	Maps input to output range
expr	Evaluate a mathematical expression

screensize

Output the monitor size

Input

bang	Triggers the output of the main screen size and total multi-monitor screen bounding rectangle out the outlets.

Arguments

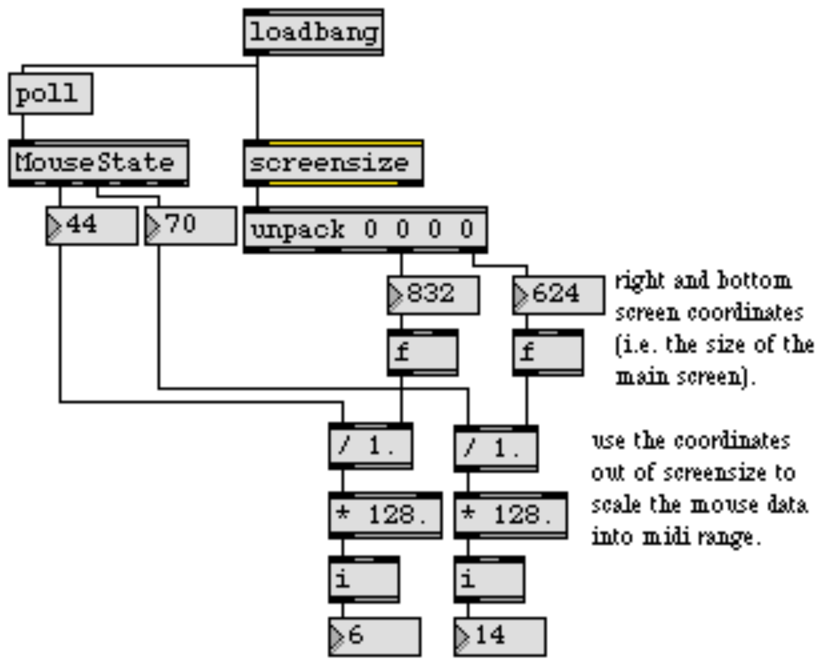
None.

Output

list Out left outlet: The bounding coordinates of the main screen: left is first, followed by top, right, and bottom.

Out right outlet: The bounding coordinates of all monitors. If there is only one monitor, the output will be the same as with the left outlet.

Examples


loadbang
poll
MouseState
screensize
44
70
unpack 0 0 0 0
832
624
right and bottom screen coordinates (i.e. the size of the main screen).
f
f
/ 1.
/ 1.
use the coordinates out of screensize to scale the mouse data into midi range.
* 128.
* 128.
i
i
6
14

screensize *reports the coordinates of the main and total screen areas*

See Also

gestalt	Inquire about current system
menubar	Put up a custom menu bar
thispatcher	Send messages to a patcher

select / sel

Select certain inputs, pass the rest on

Input

any message	In left inlet: If the input matches one of the arguments, a bang is sent out the outlet that corresponds to that argument. Otherwise, the input is passed out the right-most outlet. Note: **select** never considers an int to be a match for a float argument, or vice versa, even if their values are equal. For example, 4.0 is not considered a match for the argument 4, and 4 is not a match for 4.0.
int	In right inlet: Replaces the value of the argument. The right inlet exists only if there is a single int argument.
bang	In left inlet: Converted to symbol bang and treated as any other symbol.

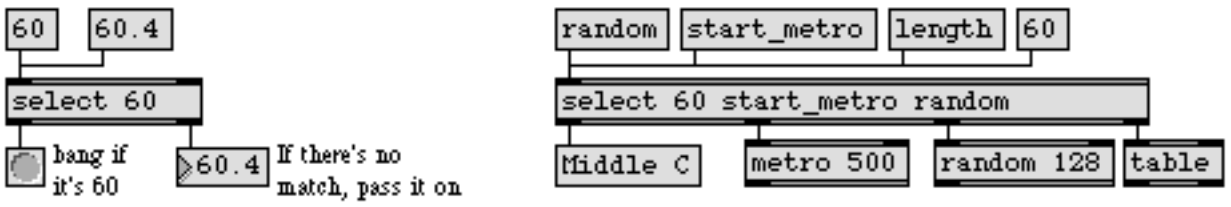
Arguments

anything	Optional. The arguments can be a mix of ints, floats, or symbols. The number of arguments determines the number of outlets in addition to the rightmost outlet. If there is no argument, there is only one other outlet, which is assigned the integer number 0.
int	If there is a single int argument (or if there are no arguments) a second inlet is created on the right. Numbers received in that inlet are stored in place of the argument. If there is more than one argument, or if the only argument is not an int, the right inlet is not created.

Output

bang	If the number or symbol received in the left inlet is the same as one of the arguments, a bang is sent out the outlet that corresponds to that argument.
anything	If the number or symbol received in the left inlet does not match any of the arguments, it is passed out the rightmost outlet.

Examples



Arguments assign names or numbers to the outlets, and a bang *is sent when the input matches them*

See Also

if	Conditional statement in if/then/else form
match	Look for a series of numbers, output it as a list
route	Selectively pass the input out a specific outlet
==	Compare two numbers, output 1 if they are equal
Tutorial 17	Gates and switches

send / s

Send messages without patch cords

Input

anything	A message received in the inlet is sent out the outlet of any **receive** object that has the same name, even if the **receive** is in another loaded patch.
(mouse)	Double-clicking on a **send** object opens all windows containing **receive** objects with the same name.

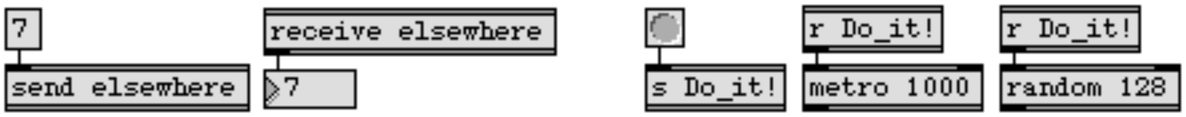
Arguments

any symbol	Obligatory. Gives a name to the **send** object.

Output

anything	There are no outlets. A message received in the inlet of **send** is sent out the outlet of any **receive** with the same name, even if the **receive** is in another loaded patch.

Examples



Virtual connections exist between all **send** *and* **receive** *objects that share the same name*

See Also

forward	Send remote messages to a variety of objects
message	Send any message
pv	Share variables specific to a patch and its subpatches
pvar	Connect to a named object in a patcher
receive	Receive messages without patch cords
value	Share a stored message with other objects
Tutorial 24	**send** and **receive**

Input

bang	Starts playing the sequence stored in **seq**.
start	The word start by itself has the same effect as bang. The word start, followed by a number, plays the stored sequence at a tempo determined by the number. The message start 1024 indicates normal tempo. If the number is 512, **seq** plays the sequence at half the original recorded speed, start 2048 plays it back at twice the original speed, and so on. The message start -1 starts the sequencer, but rather than follow Max's millisecond clock, **seq** waits for a tick message to advance its clock. See the tick message, below.
record	Starts recording MIDI messages received in the inlet.
stop	Stops the sequencer if it is recording or playing. A stop message need not be received when switching directly from playing to recording, or vice-versa.
append	Starts recording at the end of the stored sequence, without erasing the existing sequence.
int	When **seq** is recording, numbers received in its inlet are interpreted as bytes of MIDI messages (usually from **midiformat** or **midiin**). MIDI channel messages and system exclusive messages can be recorded by **seq**, but **seq** does not respond directly to MIDI real time messages such as start, stop, MIDI clock, etc.
float	Converted to int.
tick	After **seq** has received a start -1 message, it waits for tick messages to advance its clock. In order to play the sequence at its original recorded tempo, **seq** must receive 48 tick messages per second. This is equivalent to 24 ticks per quarter note (the standard for a MIDI Clock message) at a tempo of 120MM. By using tick messages to advance the sequencer, you can vary the tempo of playback or synchronize **seq** with another timing source (such as incoming MIDI Clock messages).
delay	The word delay, followed by a number, sets the onset time, in milliseconds, of the first event in the recorded sequence. All events in the sequence are shifted so that the first event occurs at the specified onset time.
hook	The word hook, followed by a float, multiplies all the event times in the stored sequence by that number. For example, if the number is 2.0, all event times will be doubled, and the sequence will play back twice as slowly. Multiplications can even be performed while the sequence is playing.
write	Calls up the standard Save As dialog box, so that a recorded sequence can be saved as a separate file. If you want to edit the sequence with the text editor, check the *Save As Text* option in the dialog box.

read — With no arguments, read calls up the standard Open Document dialog box, so that a previously recorded sequence can be read into **seq**, replacing the current sequence. With a symbol as an argument, read searches for a file with the specified name to read into the **seq** object.

Note: The **seq** object reads and writes single track (format 0) standard MIDI files. It can also read and write text files in which each line consists of a *start time* in milliseconds (the time elapsed since the beginning of the sequence) followed by the (space-separated) bytes of a MIDI message recorded at that start time. For example,

```
0 144 60 112
1000 144 60 0
1500 192 31
1500 144 60 112
2500 144 60 0
```

plays the note middle C on channel 1 for one second, then half a second later changes to program number 31 and plays middle C again for one second.

print — Prints the first sixteen events of the recorded sequence in the Max window.

dump — Opens a standard Open Document dialog box, to select a saved sequence or standard MIDI file. The selected file is opened as text in a new Untitled text window, which can be edited and saved.

Arguments

any symbol — Optional. Specifies the name of a file to be read into **seq** automatically when the patch is loaded.

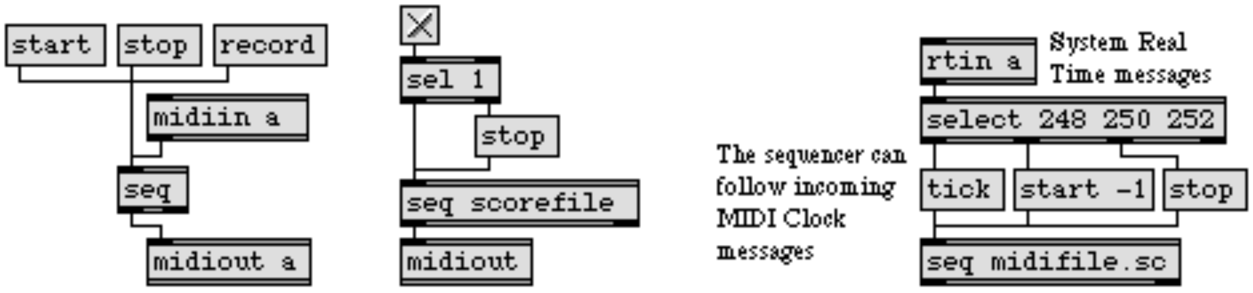
Output

int — Out left outlet: When bang or start is received in the inlet, the sequence stored in **seq** is sent out the outlet in the form of individual MIDI bytes, usually to be sent to **midiparse** or **midiout**.

bang — Out right outlet: Indicates that **seq** has finished playing the current sequence. (The bang is sent out immediately *before* the final event of the sequence is played.)

seq

Examples



Record and play back live performance, or play a pre-recorded sequence

See Also

coll	Store and edit a collection of different messages
follow	Compare a live performance to a recorded performance
mtr	Multi-track sequencer
Tutorial 35	**seq** and **follow**
Detonate	Graphic editing of a MIDI sequence
Sequencing	Recording and playing performances with MIDI

serial

Send and receive characters from serial ports and cards

The **serial** object works only with ports and devices supported by the standard serial driver. It does not work with USB ports and devices, unless a USB to Serial adaptor is connected.

Input

int	Sends the number out the serial port accessed by the **serial** object. Numbers outside the range 0-255 are wrapped to that range using a modulo operator. After the data is sent, the message write, followed by a number specifying the number of bytes sent is sent out the right outlet of the **serial** object.
list	Sends each number in the list out the serial port, in order. Numbers outside the range 0-255 are wrapped to that range using a modulo operator. After the data is sent, the message write, followed by a number specifying the number of bytes sent is sent out the right outlet of the **serial** object.
bang	Sends each character received on the serial port since the last bang message out the **serial** object's left outlet as an integer in the order that the characters were received. Before output data is sent, the message read, followed by a number specifying the number of bytes received is sent out the right outlet of the **serial** object.
bufsize	Sets the input buffer size used by the **serial** object to the value following the word bufsize. The message bufsize 0 restores the serial port's default buffer size (2048 bytes).
print	Sends a list of available serial ports to the Max window, along with their alphabetic shortcuts. The message port [portname] [portname]... is also sent from the object's right outlet, with a list of available ports.
port	The word port, followed by a symbol, specifies the serial port to be used by the object. If alphabetic shortcuts are used, a specifies the first logical serial port in the computer. b - z specify additional ports. If actual portnames are used, the symbol is the name given by the operating system to your port. See the print message, above, for a way to list available portnames and alphabetic shortcuts. If the port chosen is currently in use or unavailable when the port message is sent, an error message will be displayed and the object will revert to its previously chosen port, or won't function if there was none.
chunk	The word chunk, followed by a number that specifies list length, will cause the **serial** object to attempt to collect data into lists of that length for output. Data chunking only works if the amount of data received is greater than the chunk size—otherwise, the object will output a list as long as the available data. While chunking, the last list output may be shorter than the others if there isn't enough available data to complete the full list length.
break	Sends a break command to the serial port used by the **serial** object. After the break has completed, the message break is sent out the object's right outlet.

baud	The word baud, followed by a number that specifies a baud rate, causes the serial object to try to change the serial port baud rate. Although any integer rate is valid, the common baud rates are Some common rates are 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200, 38400 and 57600.The default is 4800 baud.
getbaud	The word getbaud will cause the **serial** object to send the message baud followed by a number that specifies the current baud rate out the **serial** object's right outlet
parity	The word parity, followed by the numbers 0, 1, or 2, or the symbols no, odd or even, causes the **serial** object to change the parity setting of the serial port used by the **serial** object. If integers are used to specify parity, 0 corresponds to no parity, 1 to odd parity, and 2 to even parity. The default is no parity (0).
getparity	The word getparity will cause the **serial** object to send the message parity, followed by a symbol that indicates the current parity (no, even, or odd) out the object's right outlet,
databits	The word databits, followed by an integer in the range 5-8, causes the **serial** object to change the number of valid data bits used while communicating with the serial port. The default value is 8.
getdatabits	The word getdatabits will cause the **serial** object to send the message databits, followed by a number in the range 5-8 that indicates the current number of databits used out the object's right outlet,
stopbits	The word stopbits, followed by the numbers 1 or 2 (or 1.5 on Windows only), causes the serial object to change the number of stop bits used when communicating with the serial port. The default value is 1.
getstopbits	The word getstopbits will cause the **serial** object to send the message stopbits followed by a number that specifies the current stopbits setting of the serial port (0, 1, or 1.5 on Windows only) out the object's right outlet.
dtr	The word dtr, followed by an integer, enables or disables the DTR (data terminal ready) function of the serial port used by the **serial** object. Non-zero integers enable the function, and 0 disables it.
getdtr	The word getdtr will cause the **serial** object to send the message dtr, followed by a number that specifies the current DTR setting of the serial port (0=disabled, 1=enabled) out the object's right outlet.

Arguments

symbol a-z or symbol portname	Optional. Specifies the serial port to be used by the **serial** object. If alphabetic shortcuts are used, a specifies the first logical serial port in the computer, and b - z are used to specify additional ports. If actual portnames are used, the symbol is the name given by the Operating System to your port. The print to the **serial** object

(see above) can be used to create a list of available portnames and alphabetic shortcuts. If the port chosen is currently in use or unavailable when the **serial** object is instantiated, an error message will be displayed and the object will not function. If no port is specified, the default port is a.

int Optional. An optional argument may be used after the port name or alphabetic shortcut to specify the baud rate of the serial port (the default rate is 4800 baud). Any value is allowable (although not all ports can be set to all baud rates). Some common rates are 300, 600, 1200, 1800, 2400, 3600, 7200, 9600, 19200, 38400 and 57600.

int Optional. After the baud rate, the next arguments specifies the number of data bits for the serial port (the default is 8 data bits). Other possible values are 5, 6 and 7.

int Optional. The next argument specifies the number of stop bits for the serial port. The default is 1. Other possible values are 1.5 (Windows only) and 2.

int or symbol Optional. The next argument specifies the parity for the serial port (the default is no parity, specified by 0 or no).Other possible values are odd, 1 (odd), even, and 2 (odd).

Output

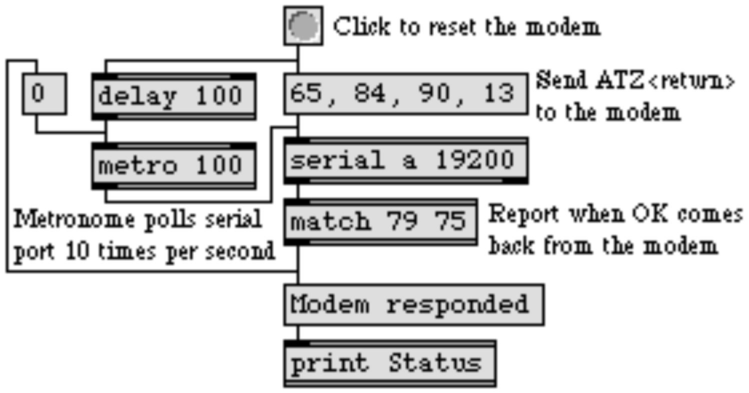
(serial output) When a number or list is received in its inlet, **serial** sends the data out the specified serial port at the current baud rate.

int When **serial** receives a bang message and characters have been received in the serial port, the received characters are sent as numbers in the order they were received.

list When **serial** receives a bang message, characters have been received in the serial port, and chunking is enabled, the received characters are sent as a list in the order the characters were received. The length of the list is determined by the argument to the chunk message (see the message listing for chunk for more information).

Out right outlet: Reports error and status messages.

Examples


Click to reset the modem
0
delay 100
65, 84, 90, 13
Send ATZ<return> to the modem
metro 100
serial a 19200
Metronome polls serial port 10 times per second
match 79 75
Report when OK comes back from the modem
Modem responded
print Status

When the button is clicked, this patch resets the modem, begins polling for a response, and stops polling when a response has been received

See Also

match	Look for a series of numbers, output it as a list
spell	Convert input to ASCII codes
vdp	Control a videodisc player through the serial port

setclock

Control the clock speed of timing objects remotely

Input

bang — In left inlet: Sends out the current time value, according to the **setclock** object's own clock. Timing objects such as **clocker**, **line**, **metro**, **pipe**, **tempo**, and **timeline** can use **setclock** as their clock source instead of Max's regular millisecond clock.

int or float — In left inlet: The meaning of the number depends on the second typed-in argument, which identifies the **setclock** object's mode of operation. If the mode is pass[ive] (the default mode), the number sets an absolute clock time which timing objects may use by comparing it to their initial time value. If the mode is add[itive], the number is added to the **setclock** object's current clock time. If the mode is interp[olate], **setclock** will change its clock time incrementally by that amount, over a time period determined by the time elapsed since the previous number was received. (However, negative numbers cause an *immediate* decrease in the clock time.) If the mode is ext[ernal] or mul[tiplicative], the number is simply ignored. If the mode is mul[tiplicative], the number is used as a multiplier for associated timing objects. For instance the number 0.5 halves the rate of increase (speed) of the associated timing objects. If the mode is ext[ernal], the number is ignored.

In right inlet: Sets the time interval, in milliseconds, at which the **setclock** will report its clock information to associated timing objects. The default is 5 milliseconds.

set — If the **setclock** is in pass[ive] or add[itive] mode, the word set followed by a number sets its clock time to that number. If **setclock** is in any other mode, the set message is ignored.

reset — If **setclock** is in interp[olate] mode, the word reset followed by a number sets its clock time to that number, then repeats the last interpolation it performed.

Arguments

any symbol — Obligatory. The first argument is the *name* of the **setclock** object, by which timing objects such as **clocker**, **line**, **metro**, **pipe**, **tempo**, and **timeline** can refer to the **setclock**. Those timing objects—once they have received the message clock followed by the name of a **setclock** object—use that **setclock** as their timing source instead of Max's regular millisecond clock. The **setclock** object need not be in the same patcher as the timing objects that refer to it. More than one **setclock** object may exist with the same name; **setclock** objects with the same name share the same clock time information. (Note: Different **setclock** objects that share the same name argument can have different mode arguments typed in, but they will in fact operate with the mode of whichever **setclock** was *first* loaded with that name. Thus, **setclock** objects with the same name but different modes may behave unpredictably, since the order in which they are loaded by Max is often unknown.)

The second (optional) argument describes the *mode* of clock operation this **setclock** object will have. The possible modes for the second argument are:

pass Specifies *passive* mode. In this mode, the **setclock** object's current clock time is set by a number received in the left inlet, and associated timing objects will follow that clock time just as if it were a regularly progressing millisecond clock. If no second argument is present, the mode is pass by default.

add Specifies *additive* mode. A number received in the left inlet is added to the current clock time to determine the new clock time.

mul Specifies *multiplicative* mode. The number received in the left inlet is used as a factor by which all associated timing objects will modify their time settings. For example, a factor of 2.0 will cause all timing objects that are using the **setclock** as their clock source to double their time values (that is, to halve their speed). An alternative (and perhaps more truthful) way to conceptualize the behavior of mul mode is to think of the incoming float as a divisor by which **setclock** divides the speed at which its own clock time progresses. Thus, when it receives the number 2.0 it divides its own clock speed by 2.0, causing the objects which are following that clock to progress twice as slowly.

interp Specifies *interpolate* mode. The number received in the left inlet is gradually added to the current time of **setclock**, over a time period determined by the amount of time elapsed since the *previous* number was received. During that time period, **setclock** linearly interpolates to set its clock to the intermediate values.

float If the second argument is mul, an optional third argument specifies a multiplier for the time of all associated timing objects. If no third argument is present, the multiplier is 1.0 by default.

Additional possible modes for the second argument are:

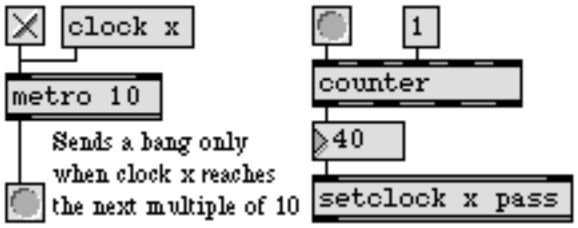
Output

int When bang is received in the left inlet, **setclock** sends its current time reading out the outlet.

setclock

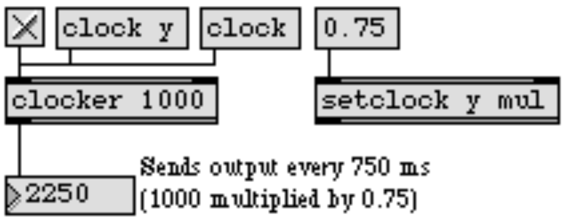
Control the clock speed of timing objects remotely

Examples



clock x
metro 10
Sends a bang only when clock x reaches the next multiple of 10
1
counter
40
setclock x pass

setclock *becomes the clock for* **metro**



setclock *modifies the time for* **clocker**

See Also

clocker	Report elapsed time, at regular intervals
metro	Output a bang message at regular intervals
timeline	Time-based score of Max messages
timer	Report elapsed time between two events
Timeline	Creating a graphic score of Max messages

sin

Input

float or int	Input to a sine function in radians.

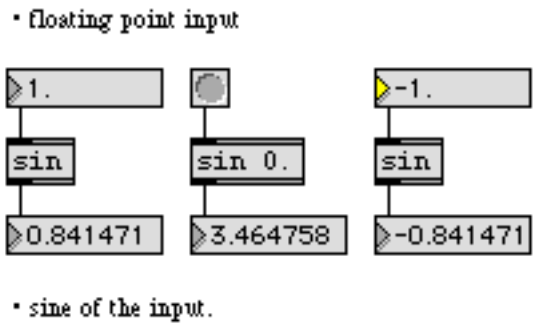
Arguments

float or int	Optional. Sets the initial value for the sine function.

Output

float or int	The sine of the input in radians.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sinh	Hyperbolic sine function
tan	Tangent function
tanh	Hyperbolic tangent function

sinh

Input

float or int	Input to a hyperbolic sine function.
bang	In left inlet: Calculates the hyperbolic sine of the number currently stored. If there is no argument, **sinh** initially holds 0.

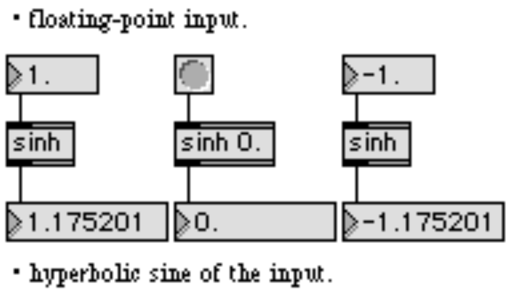
Arguments

float or int	Optional. Sets the initial value for the hyperbolic sine function.

Output

float or int	The hyperbolic sine of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
tan	Tangent function
tanh	Hyperbolic tangent function

slider

Output numbers by moving a slider onscreen

Input

int	The number received in the inlet is displayed graphically by **slider**, and is passed out the outlet. Optionally, **slider** can multiply the number by some amount and add an offset to it, before sending it out the outlet.
(mouse)	The **slider** will also send out numbers in response to dragging on it directly with the mouse.
float	Converted to int.
bang	Sends out the number currently stored in the **slider**.
min	The word min, followed by a number, sets a value that will be added to the **slider** object's value before it is sent out the outlet. The default is 0.
mult	The word mult followed by a number, specifies a multiplier value. The **slider** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default value is 1.
set	The word set, followed by a number, resets the value displayed by the **slider**, without triggering output.
size	The word size, followed by a number, sets the range of the **slider** object. The default value is 128.

Inspector

The behavior of a **slider** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **slider** object displays the **slider** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **slider** Inspector lets you enter a *Slider Range* value. Numbers received in the inlet are automatically limited between 0 and the number 1 less than the specified range value. The default range value is 128. You can specify an *Offset* value which will be added to the number, after multiplication. The default offset value is 0. The **slider** Inspector also lets you specify a *Multiplier*. The **slider** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default multiplier value is 1.

slider

Output numbers by moving a slider onscreen

Arguments

The range of **slider** is set by selecting it (when the patcher window is unlocked) and choosing Get Info… from the Object menu. The **slider** automatically resizes itself to accommodate the new range.

The Inspector also provides a *Multiplier*—by which all numbers will be multiplied before being sent out, and an *Offset*—which will be added to the number, after multiplication. A newly created **slider** has a range of 128, a multiplier of 1, and an offset of 0.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Output

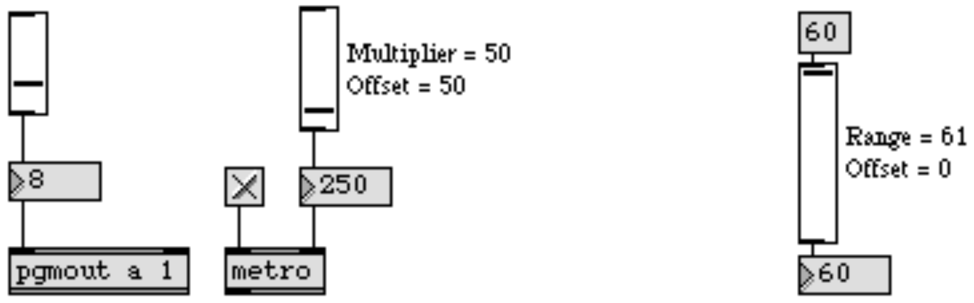
int Numbers received in the inlet, or produced by dragging on **slider** with the mouse, are first multiplied by the multiplier, then have the offset added to them, then are sent out the outlet.

Although the numbers that can be output by dragging are limited by the range of the **slider**, numbers received in the inlet are not limited before they are sent out the outlet.

slider

Output numbers by moving a slider onscreen

Examples



Produce output by dragging onscreen... *or display numbers passing through*

See Also

dial	Output numbers by moving a dial onscreen
hslider	Output numbers by moving a slider onscreen
kslider	Output numbers from a keyboard onscreen
multislider	Multiple slider and scrolling display
pictctrl	Picture-based control
pictslider	Picture-based slider
rslider	Display or change a range of numbers
uslider	Output numbers by moving a slider onscreen
Tutorial 9	Using the slider
Tutorial 14	Sliders and dials

speedlim

Limit the speed with which messages can pass through

Input

anything	In left inlet: The message is passed out the outlet, provided that a certain minimum time has elapsed since the previous output. Otherwise, the message is held until that amount of time has passed (or until it is overwritten by another incoming message).
int	In right inlet: The number is stored as the minimum amount of time, in milliseconds, between successive outputs.
clock	In left inlet: The word clock, followed by the name of an existing **setclock** object, causes the time interval of **speedlim** to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets **speedlim** back to using Max's regular millisecond clock.

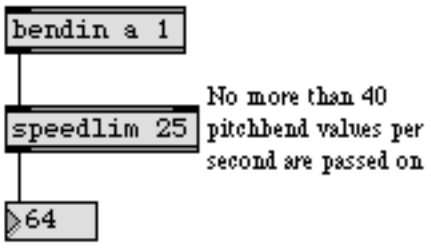
Arguments

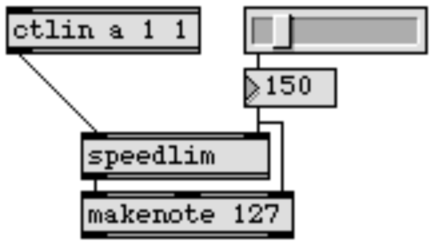
int	Optional. Sets an initial minimum time between outputs, in milliseconds. If there is no argument, the minimum time is 0.

Output

anything	A message received in the left inlet is sent out the outlet, provided the specified minimum amount of time has elapsed since the previous output. Otherwise, **speedlim** waits until that amount of time has passed, then sends out the last message it has received since the previous output.

Examples





Used to reduce a heavy flow of numbers, or to turn a continuous flow into discrete steps

See Also

delay	Delay a bang before passing it on
mousefilter	Pass numbers only when the mouse button is up
thresh	Combine numbers into a list, when received close together
timer	Report elapsed time between two events
Tutorial 16	More MIDI ins and outs

spell

Convert input to ASCII codes

Input

any symbol — The ASCII value of each letter, digit, or other character in the symbol is sent out the outlet, one character at a time.

int — The ASCII value of each of the digits of the number is sent out the outlet, one digit at a time.

list — Each int in the list is converted to ASCII as described above, and a space character (32) is sent out between items in the list. Any float or symbol items in the list are ignored.

any message — If the message begins with a symbol, all int and symbol items in the message are converted to ASCII one character at a time, and a space character (32) is placed between them. Any float items in the list are ignored. If the message begins with a float, both floats and symbols are ignored.

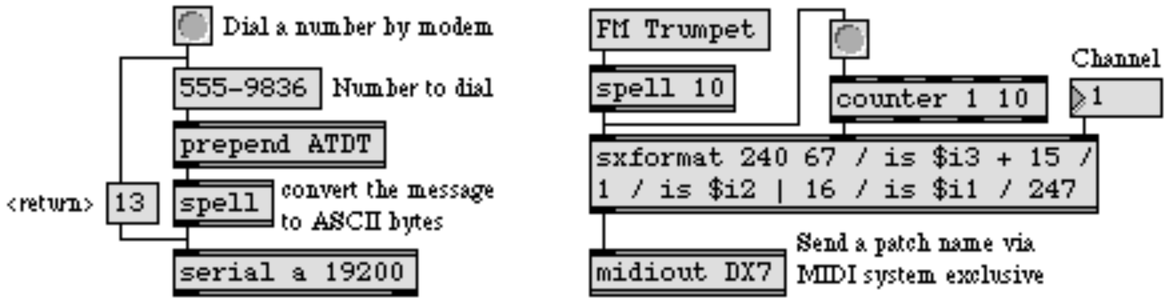
Arguments

int — Optional. The first argument sets the minimum output size. Any input that doesn't "spell" to the minimum length is followed by enough fill characters (the default is the space character, 32 in ASCII) to satisfy the minimum requirement. A second optional argument specifies the fill character to use instead of 32. If you want to use '0' as a fill character, use any negative number as a second argument to **spell**.

Outputs

int — The ASCII representation of the input is sent out one character at a time.

Examples


Dial a number by modem
555-9836
Number to dial
prepend ATDT
<return>
13
spell
convert the message to ASCII bytes
serial a 19200
FM Trumpet
spell 10
counter 1 10
Channel
1
sxformat 240 67 / is $i3 + 15 / 1 / is $i2 | 16 / is $i1 / 247
midiout DX7
Send a patch name via MIDI system exclusive

*Using the **spell** object, a modem command string or a synthesizer patch name can be translated from human terms into computer terms, and sent out the serial port in ASCII representation*

See Also

key	Report key presses on the computer keyboard
keyup	Report key releases on the computer keyboard
message	Send any message
sprintf	Format a message of words and numbers

split

Input

int or float — In left inlet: If the number is within a specified range, it is sent out the left outlet. Otherwise, it is sent out the right outlet.

In middle inlet: The number is stored as the minimum value in the range of numbers looked for by **split**. If the number is an int, then the **split** object will convert all float values to ints.

In right inlet: The number is stored as the maximum value in the range of numbers looked for by **split**.

list — In left inlet: The second number is stored as the minimum value of the range, and the third number is stored as the maximum value of the range. The first number is then compared to the range, and is sent out one of the two outlets.

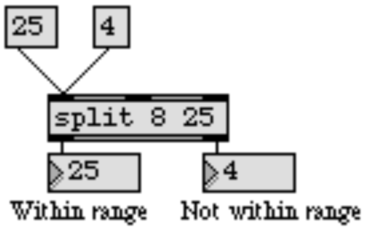
Arguments

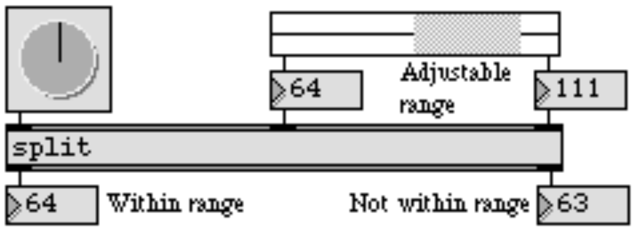
int or float — Optional. The first argument sets the minimum value to be sent out the left outlet. If the first argument is an int, then the **split** object will convert all float values to ints. The second argument sets the maximum value to be sent out the left outlet. If the first argument to split is an int, the output is int. If it is float, the output is float. This is true regardless of the type of the input.

Output

int — If the number received in the left inlet is *greater than or equal to* the specified minimum, and it is *less than or equal to* the specified maximum, it is sent out the left outlet. Otherwise, it is sent out the right outlet.

Examples





Used to divert a certain range of numbers to a different destination

See Also

route	Selectively pass the input out a specific outlet
select	Select certain inputs, pass the rest on
<=	*Is less than or equal to,* comparison of two numbers
>=	*Is greater than or equal to,* comparison of two numbers
Tutorial 20	Using the computer keyboard

spray

Distribute an integet to a numbered outlet

Input

list	The first number in the list specifies the outlet number; the second is the number to send out that outlet. If there are additional numbers in the list, they are sent out the subsequent outlets to the right of the one specified by the first number in the list. The list may contain only ints; floats (or symbols) will be ignored.

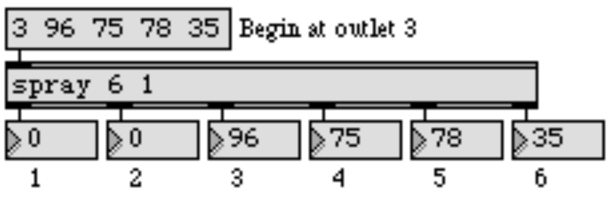
Arguments

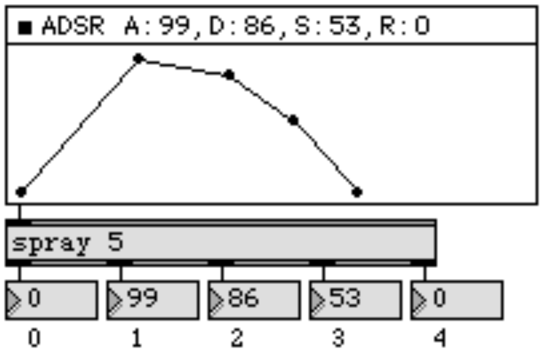
int	Optional. The first argument sets the number of outlets. If there is no argument present, the object has two outlets. The second argument sets an offset for the numbering of the outlets. If the second argument is not present, the outlets are numbered beginning with 0.

Output

int	When a list of ints is received by **spray**, the first number is used to specify an outlet, and the second number is sent out that outlet. Any additional numbers in the list are sent out subsequent outlets to the right. You can connect the outlet of an **env** or **envi** object to the inlet of a **spray** object to distribute the envelope's values to separate outlets.

Examples





Used to break up a list and send the items out specific outlets

See Also

cycle	Send a stream of data to individual outlets
env	Script-configurable envelope editor
envi	Script-configurable envelope in a patcher window
funnel	Map a number to a list which identifies its inlet
gate	Pass the input out a specific outlet
route	Selectively pass the input out a specific outlet
unpack	Break a list up into individual messages

sprintf

Format a message of words and numbers

Input

int — May be received in any inlet that corresponds to a %ld or %c argument. The number will be stored in place of that argument. A %c argument will convert the int to its ASCII character equivalent.

float — May be received in any inlet that corresponds to a %f argument. The number will be stored in place of that argument.

symbol — May be received in any inlet that corresponds to a %s argument. The number will be stored in place of that argument.

list — In left inlet: Each item in the list is treated as if it had been received in a separate inlet, up to the number of inlets.

bang — In left inlet: Formats the message using the values currently stored.

Any of the above messages in the left inlet will format the message and send it out. If no value has been received for a changeable number argument (%ld or %f), 0 will be substituted for that argument. If no value has been received for a %s or %c argument, that argument will be left blank.

Arguments

symout — Optional. If the first argument is the word symout, the **sprintf** object outputs the string it generates as a single symbol. Otherwise the output is a list of symbols and/or numbers. The word symout itself is not included in the output of sprintf.

Obligatory. The arguments form a message to be sent out, in a format resembling the C programming language. The arguments may be words, numbers, or changeable arguments for incoming symbols (%s), ints (%ld), floats (%f), and ints that are to be formatted as ASCII characters (%c). The number of inlets is determined by the number of changeable arguments, with each inlet corresponding to a changeable argument, in order.

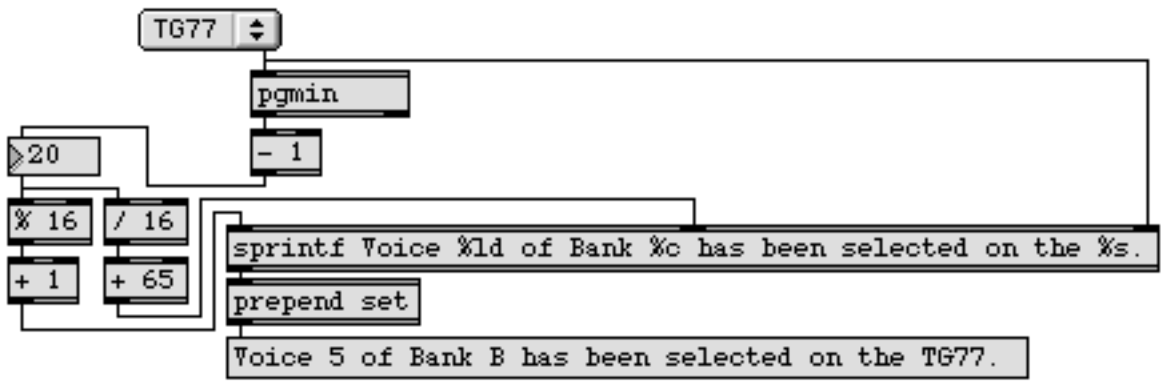
Output

anything — The message specified by the typed-in argument(s) is formatted and sent out with substitutions made for the changeable arguments.

sprintf

Format a message of words and numbers

Examples



TG77
pgmin
20
- 1
% 16
/ 16
+ 1
+ 65
sprintf Voice %ld of Bank %c has been selected on the %s.
prepend set
Voice 5 of Bank B has been selected on the TG77.

Changeable arguments are replaced by values received in the inlets.

See Also

fromsymbol	Transform a symbol into individual numbers or messages
key	Report key presses on the computer keyboard
keyup	Report key releases on the computer keyboard
message	Send any message
spell	Convert input to ASCII codes
tosymbol	Convert messages, numbers, or lists to a single symbol

sqrt

Square root function

Input

int or float	**sqrt** outputs the square root of the input value. A negative input has no real solution, so it causes an output of NaN (Not a Number).
bang	Outputs the currently stored square root value.

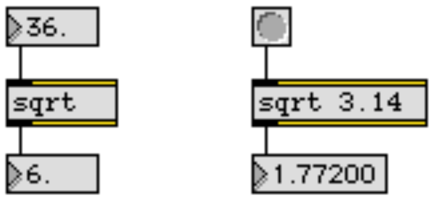
Arguments

int or float	Optional. An optional argument specifies the value whose square root is to be output.

Output

float	The square root of the input.

Examples



See Also

expr	Evaluate a mathematical expression

standalone

Note: Building standalone applications is not currently supported on Windows. The **standalone** object and its Inspector are included with the Windows release of Max, but will have no function until support for standalone applications is added to the Windows version.

The **standalone** object lets you set options for creating a standalone application from a Max/MSP patch, and is used in conjunction with the Build Application/Collective... item found in the Edit menu. You should only have one **standalone** object in your top-level patch.

Input

All parameters for standalone applications are set using the **standalone** object's Inspector.

Inspector

The behavior of a **standalone** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting the **standalone** object displays the **standalone** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **standalone** Inspector lets you set the following attributes:

The *Application Creator Code* is a four-letter file type code that will endow your standalone application with a personal identity in the computer's file system.

The File Options section lets your application have its own "plist" resource if you check the

Checking the *Use Own Property List (plist) Resource* option lets your application have its own "plist" resource and makes it possible for you to customize your standalone application's icons (analogous to the BNDL resource on OS9).

If some of the supporting files used by Max/MSP objects in your patch will not be included in the collective itself, check the *Search for Files Not in the Application's Collective* option.

Checking the *Utilize Search Path in Preferences File* option lets you use the search path stored in the Preferences file instead of using the default search path.

If you want to use your own preferences file to save default settings for your standalone application instead of the default Max Preferences file, you can specify the file name in the *Preference File Name* box.

The *Options* section lets you set the behavior and appearance of the standalone application.

Checking the *Status Window Visible at Startup* option will display a Status window (similar to the Max window) when you launch the Standalone application.

To assure that users of your standalone application can't abort the loadbang message sent to all objects when the top-level patch is loaded, check the *Prevent Loadbang Defeating with Cmd-Shift* option.

You can enable the standard Max/MSP Overdrive and All Windows Active behavior in your standalone application by checking the *Overdrive Enabled* and *All Windows Active Enabled* options.

You can keep users from being able to close the top-level patcher in your standalone application by checking the *User Can't Close top-level Patcher Windows* option.

Arguments

None.

Output

None.

See Also

Collectives	Grouping files to create a single application.

stripnote

Filter out note-off messages, pass only note-on messages

Input

list	In left inlet: The second number is stored as a velocity, and the first number is treated as the pitch, of a MIDI note-on message. If the second number is not 0, it is sent out the right outlet, and the first number is sent out the left outlet. If the second number is 0, nothing is sent out.
int	In left inlet: The number is treated as a pitch value. If the velocity value currently held by **stripnote** is not 0, then the velocity is sent out the right outlet and the pitch is sent out the left outlet.
	In right inlet: The number is stored as a velocity to be paired with pitch numbers received in the left inlet.
float	Converted to int.

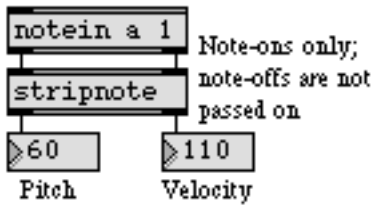
Arguments

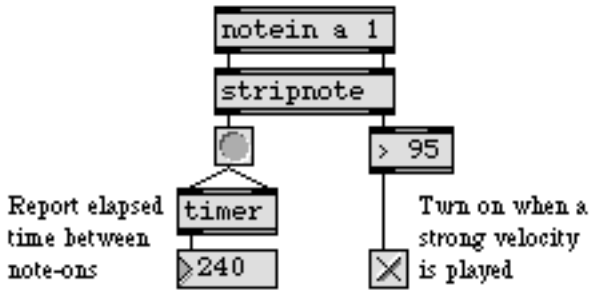
None.

Output

int	Out left outlet: The pitch value received in the left inlet is sent out, provided the velocity is not 0.
	Out right outlet: The velocity value of a note-on pair is sent out, provided it is not 0.

Examples




notein a 1
stripnote
> 95
Report elapsed time between note-ons
timer
240
Turn on when a strong velocity is played

Repeated pitch values and 0 velocities caused by note-off messages can be filtered out

See Also

makenote	Generate a note-off message, following each note-on
sustain	Hold note-off messages, output them on command
Tutorial 13	Managing note data

strippath

Get a filename from a full pathname

Input

symbol	An absolute pathname as a symbol. An absolute pathname looks like this: "MyDisk:/Max Folder/extras/filename"

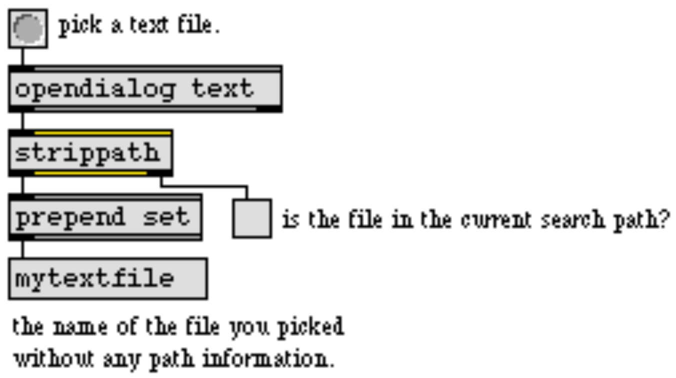
Arguments

None.

Output

symbol	Out left outlet: The file name, with all path information preceding it removed.
int	Out right outlet: If the file was found within the current Max search path a 1 is sent out the right outlet. A 0 is sent otherwise.

Examples



pick a text file.
opendialog text
strippath
prepend set
is the file in the current search path?
mytextfile
the name of the file you picked without any path information.

strippath *removes path information from a file pathname, and leaves you the name of the file*

See Also

absolutepath	Convert a file name to an absolute path
conformpath	Convert paths of one pathtype and/or pathstyle to another
dropfile	Define a region for dragging and dropping a file
opendialog	Open a dialog to ask for a file or folder
relativepath	Convert an absolute to a relative path
savedialog	Open a dialog to ask for a filename for saving

substitute

Subsitute a symbol for another symbol in a message

Input

anything	In left inlet: The input is echoed to the output, but if the message received contains an element matching the match symbol or number, the element is replaced by the replacement symbol or number when the message is repeated to the output.
anything	In right inlet: The **substitute** object accepts a message of two numbers or symbols in its right inlet. The first number or symbol specifies the match, which identifies what should be replaced in an incoming message.
set	In right inlet: Same as anything, except that the word set is ignored.

Arguments

anything	Optional. The first number or symbol specifies the match, which identifies what should be replaced in an incoming message. The default match value is 0.
anything	Optional. The second number or symbol specifies the replacement for the match. The default replacement value is 0.
anything	Optional. The second number or symbol specifies the replacement for the match. The default replacement value is 0.
anything	Optional. Any third number or symbol sets the "replace message only" mode of the **substitute** object. Only the first instance of the specified match will be replaced.

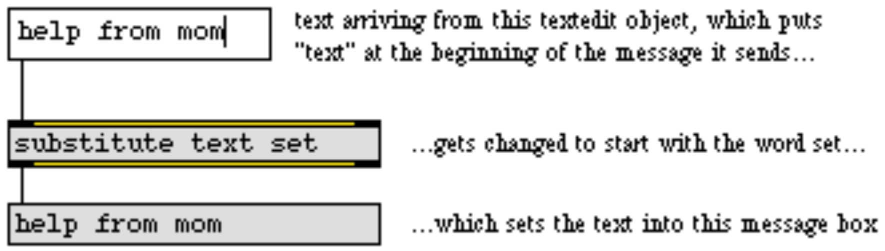
Output

anything	Out left outlet: The input message is echoed to the output with elements matching the match symbol or number replaced by the replacement number or symbol.
bang	Out right outlet: If no substitution occurred when sending out the incoming message, a bang is sent.

substitute

Subsitute a symbol for another symbol in a message

Examples



help from mom
text arriving from this textedit object, which puts "text" at the beginning of the message it sends…
substitute text set
…gets changed to start with the word set…
help from mom
…which sets the text into this message box

substitute *can translate messages output by one object to what's expected by another object*

See Also

route	Selectively pass the input out a specific outlet
sprintf	Format a message of words and numbers
zl	Multi-purpose list processor

suspend

Reports when application is suspended and resumed

Input

None.

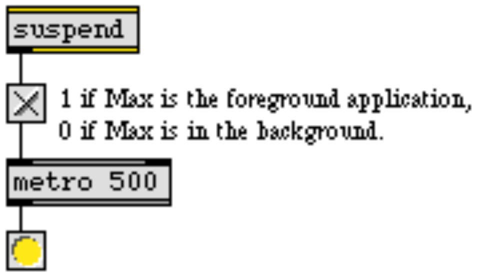
Arguments

None.

Output

int	Out left outlet When the application is suspended (made to go into the background), a 1 is output. When the application is resumed (restored to being in the foreground), a 0 is output.

Examples


suspend
1 if Max is the foreground application,
0 if Max is in the background.
metro 500

suspend *lets you activate/deactivate processes if Max is the foreground application*

See Also

active	Send 1 when patcher window is active, 0 when inactive
gestalt	Inquire about current system

sustain

Hold note-off messages, output them on command

Input

list	In left inlet: The second number is stored as the velocity, and the first number is treated as the pitch, of a MIDI note-on message. If the pair is a note-on (the velocity is not 0), the velocity is sent out the right outlet and the pitch is sent out the left outlet. Note-offs (note-ons with a velocity of 0) are either passed on immediately or held by **sustain**.

int	In left inlet: The number is the pitch value of a pitch-velocity pair. If the velocity value currently held by **sustain** is not 0, then the pair is sent out immediately. If the velocity is 0, the note-off is either sent out or held, depending on whether **sustain** is turned on.

In middle inlet: The number is stored as a velocity to be paired with pitch numbers received in the left inlet.

In right inlet: If the number is not 0, **sustain** is turned on, and all *note-offs* are held. If the number is 0, **sustain** is turned off, and all note-offs are sent out immediately.

float	Converted to int.

Arguments

None.

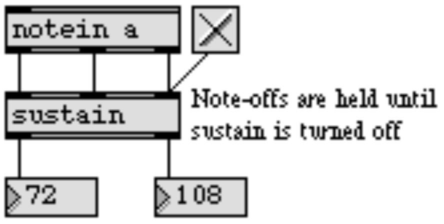
Output

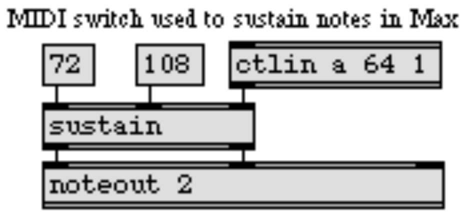
int	Out left outlet: The pitch value of a pitch-velocity pair.

Out right outlet: The velocity value of a pitch-velocity pair.

Note-on pairs are always sent out immediately. If **sustain** is turned on, note-offs are held until it is turned off. Otherwise, note-offs are sent out immediately.

Examples





Like the sustain pedal of a piano, **sustain** *releases all held notes at one time*

See Also

flush	Provide note-offs for held notes
makenote	Generate a note-off message, following each note-on
stripnote	Filter out note-off messages, pass only note-on messages

swap

Reverse the sequential order of two numbers

Input

int	In left inlet: The number is sent out the right outlet, then the number in the right inlet is sent out the left outlet. In right inlet: The number is stored to be sent out the left outlet when a number is received in the left inlet.
float	The numbers are converted to int, unless there is a float argument, in which case the number received in the right inlet is stored as a float.
list	In left inlet: The numbers are stored in **swap**. The first number is sent out the right outlet, then the second number is sent out the left outlet.
bang	In left inlet: Swaps and sends out the numbers currently stored in **swap**.

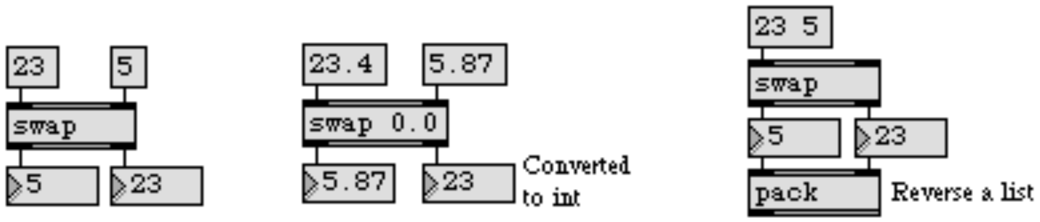
Arguments

int or float	Optional. Sets an initial value for the number that is to be sent out the left outlet. Float argument will cause a float to be sent out the left outlet. (The number sent out the right outlet is always an int.) If there is no argument, the initial value is 0.

Output

int	When a number is received in the left inlet, the number in each inlet is sent out the opposite outlet.
float	If there is a float argument, the number sent out the left outlet is a float.

Examples



Numbers are sent out in reverse order from that in which they were received

See Also

buddy	Synchronize arriving data, output them together
fswap	Reverse the sequential order of two decimal numbers
pack	Combine numbers and symbols into a list
unpack	Break a list up into individual numbers
Tutorial 30	Number groups

swatch

Color swatch for RGB color selection and display

The 2-dimensional colorspace of the **swatch** object represents hue along the horizontal axis, and luminance along the vertical axis. a third color dimension, saturation, may be set by means of the saturation message.

Input

int	In left inlet: A number between 0 and 255 sets the red color component and causes output. In middle inlet: A number between 0 and 255 sets the green color component and causes output. In right inlet: A number between 0 and 255 sets the blue color component and causes output. Note: Unlike most Max objects, input to any one of the three inlets will re-calculate the current color location on the swatch, and trigger output).
float	Converted to int.
(mouse)	Clicking and dragging on the **swatch** will calculate and output the RGB color at the selected (*x*, *y*) position on the 2-dimensional (hue-luminance) colorspace, taking into account the current saturation value.
bang	causes output of the RGB values of the current color at the selected (*x*, *y*) position on the 2-dimensional colorspace, taking into account the current saturation value.
hsl	The word hsl, followed by a list of three numbers between 0 and 255, sets the color based on the given hue (x-axis), saturation, and luminance (y-axis) values, The **swatch** object converts these values to RGB color values, refreshes the display and causes output of the RGB values.
list	a list of three numbers between 0 and 255 sets the three RGB color components (red, green, blue), refreshes the display and causes output.
saturation	the word saturation, followed by a number between 0 and 255 will change the color saturation of the displayed 2-dimensional (hue, lightness) colorspace It will also re-calculate the new RGB color at the selected (*x*, *y*) position and cause output.
set	The word set, followed by a list of three numbers between 0 and 255 sets the three RGB color components (red, green, blue) and refreshes the display without causing output.
sethsl	The word sethsl, followed by a list of three numbers between 0 and 255, sets the color based on the given hue (x-axis), saturation, and luminance (y-axis) values and the refreshes the display. Unlike the hsl message the sethsl message does not output the corresponding RGB values.

(preset)	You can save and restore the **swatch** object's RGB color using a **preset** object.

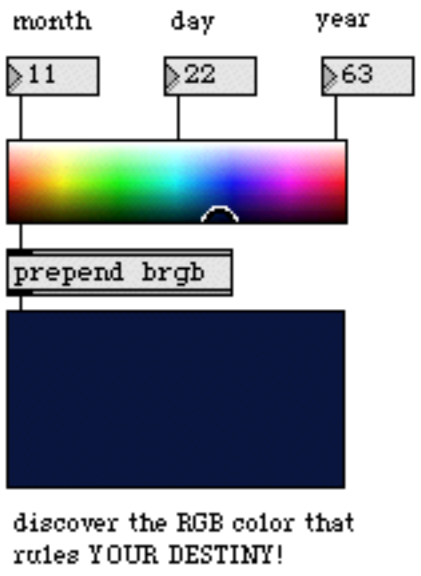
Arguments

None.

Output

list	Out left outlet: a list of three RGB (red, green, blue) color values
int	Out right outlet: the current saturation value (calculated from an RGB list input, or output directly after a saturation message)

Examples



Display input RGB values

See Also

colorpicker	Select a color using a modal dialog
panel	Colored background area

switch

Output messages from a specific inlet

Input

int	In left inlet: The number specifies an open inlet for receiving subsequent messages to be sent out the outlet. All inlets other than the designated open one are closed. If the number is 0, all inlets are closed.
anything	In any other inlet: Any message received in an *open* inlet is passed out the outlet. Messages received in closed inlets are ignored.
float	In left inlet: Converted to int.
bang	In left inlet: Sends out the number of the open inlet, or 0 if all inlets are closed.

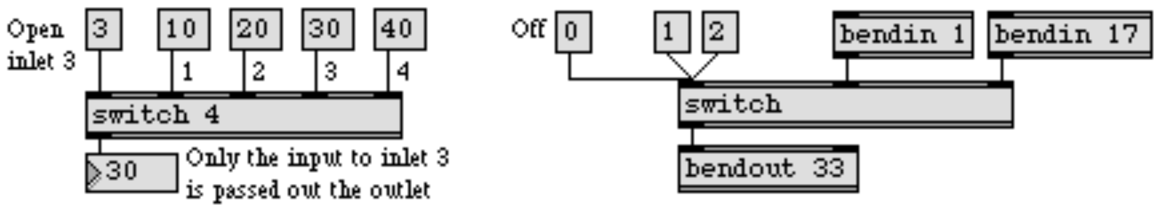
Arguments

int	Optional. Specifies the number of inlets, up to 10, *in addition to* the leftmost inlet. If there is no argument, there are two additional inlets.

Output

anything	If the number in the left inlet is less than 0, its absolute value is used to determine which inlet to open. (-1 opens inlet 1, -2 opens inlet 2, etc.) If the absolute value of the number is greater than the number of existing inlets, messages are received in the rightmost inlet.

Examples



"Listen" to only one inlet at a time, or ignore all inlets

See Also

forward	Send remote messages to a variety of objects
funnel	Tag data with a number that identifies its inlet
gate	Pass the input out a specific outlet
Ggate	Pass the input out one of two outlets
Gswitch	Receive the input in one of two inlets
receive	Receive messages without patch cords
send	Send messages without patch cords
Tutorial 17	Gates and switches

sxformat

Prepare MIDI system exclusive messages

Input

int	In left inlet: The number replaces any $i1 arguments in the object box, and the entire list of arguments is evaluated and sent out the outlet, one-by-one.
	In other inlets: The number is stored in place of the $i argument that corresponds to that inlet, until a number is received in the left inlet.
list	In left inlet: The numbers in the list are used to replace the corresponding $i arguments in the object box, then the list of arguments is evaluated and the numbers are sent out one-by-one.
bang	In left inlet: Sends out the bytes of the formatted message, using the most recently received numbers.

Arguments

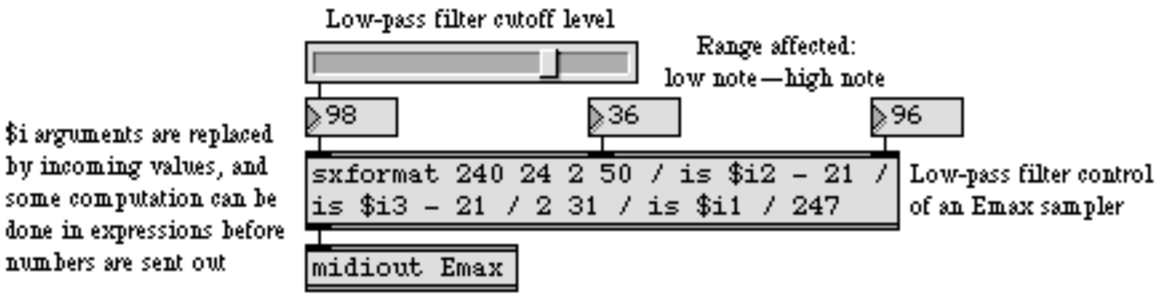
list	Obligatory. The arguments are a list of numbers which represent the values of individual bytes of a MIDI system exclusive message. The first number should be 240 (or 0xF0), the system exclusive status byte and the last number should be 247 (or 0xF7), the end byte. There can be any number of values for data bytes in between.
	Arguments for data bytes can also be in the form of a mathematical expression (like the expressions in **expr** and **if** objects) to be evaluated before numbers are sent out the outlet. The expressions can contain changeable arguments in the form $i, followed immediately by an inlet number (for example, $i2). The changeable arguments are replaced by numbers received in the specified inlet. Expressions used in place of numbers should be preceded by the word is, and should be separated from other arguments with a slash (/) on either side of the expression (see example).
	If the value of an evaluated expression is less than 0, no number is sent out in place of that expression. This allows you to send variable-length system exclusive messages.

Output

int	When a number is received in the left inlet, any expressions in the argument are evaluated and the numbers in the list are sent out one at a time, as bytes of a MIDI system exclusive message, for transmission by **midiout**.

sxformat

Examples


Low-pass filter cutoff level
Range affected: low note—high note
98
36
96
$i arguments are replaced by incoming values, and some computation can be done in expressions before numbers are sent out
sxformat 240 24 2 50 / is $i2 - 21 / is $i3 - 21 / 2 31 / is $i1 / 247
Low-pass filter control of an Emax sampler
midiout Emax

sxformat *can send a complete MIDI system exclusive message, byte-by-byte, to* **midiout**

See Also

expr	Evaluate a mathematical expression
midiout	Transmit raw MIDI data
sysexin	Output received MIDI system exclusive messages
Tutorial 34	Managing raw MIDI data
MIDI	MIDI overview and specification

sysexin

Output received MIDI system exclusive messages

Input

(MIDI)	The **sysexin** object receives MIDI system exclusive messages from a MIDI input device.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming MIDI messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **sysexin** object shows a pop-up menu for choosing a MIDI port or device.

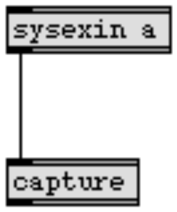
Arguments

a-z	Optional. Specifies the port from which to receive incoming MIDI system exclusive messages. If there is no argument, **sysexin** receives from port a (or the first input port listed in the **MIDI Setup** dialog.)

Output

int	MIDI system exclusive messages received from the specified port are sent out the outlet, byte-by-byte.

Examples



Examine incoming System Exclusive messages

See Also

midiin	Output received raw MIDI data
sxformat	Prepare MIDI system exclusive messages
Tutorial 34	Managing raw MIDI data
MIDI	MIDI overview and specification
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified

table

Store and graphically edit an array of numbers

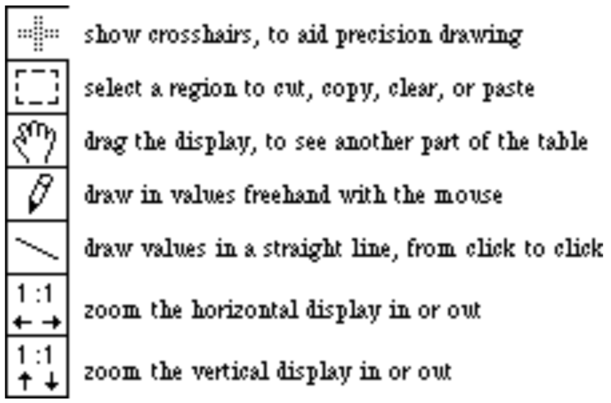
Input

list	In left inlet: The second number is stored in **table**, at the address (index) specified by the first number.
int	In left inlet: The number specifies an address in the **table**. The value stored at that address is sent out the left outlet. *However,* if a value has been received in the right inlet, **table** stores that value in the specified address, rather than sending out a number.
	In right inlet: The number specifies a value to be stored in **table**. The next address number received in the left inlet causes the value to be stored at that address.
float	Converted to int.
bang	In left inlet: Same as a quantile message with a random number between 0 and 32,768 as an argument.
cancel	In left inlet: Causes **table** to forget a number received in the right inlet, so that the next number received in the left inlet will send out a number, rather than storing a number at that address.
clear	In left inlet: Sets all values in the **table** to 0.
const	In left inlet: The word const, followed by a number, stores that number at all addresses in the **table**.
dump	In left inlet: Sends all the numbers stored in the **table** out the left outlet in immediate succession, beginning with address 0.
flags	In left inlet: Changes the **table** object's saving options, which can be found in the Inspector (see above). The word flags is followed by two number arguments. The first argument affects the *Save with patcher* option, and the second argument affects the *Don't Save* option. If the argument is non-zero the option is checked; if the argument is 0 the option is unchecked. For example, the message flags 1 1 will cause the **table** object's contents to be saved as part of the patch that contains it, and Max will not ask to save any changes that are made to the **table**.
fquantile	In left inlet: The word fquantile, followed by a number between 0 and 1, multiplies the number by the sum of all the numbers in the **table**. Then, **table** sends out the address at which the sum of the all values up to that address is greater than or equal to the result.
getbits	In left inlet: Gets the value of one or more specific bits of a number stored in the **table**, and sends that value out the left outlet. The word getbits is followed by three number arguments. The first argument is the address being referred to; the second argument is the starting bit location in the number stored at that address (the bit locations are numbered 0 to 31, from the least significant bit to the most signif-

	icant bit); and the third argument specifies how many bits to the right of the starting bit location should be sent out. The specified bits are sent out the outlet as a single decimal integer.
	For example, the message getbits 61 4 3 will look at address 61 in the **table**, start at bit location 4 (the fifth bit from the right), and send out the decimal number that corresponds to the 3 bits starting at that location. So, suppose that address 61 of the table stores the number 87. The binary form of 87 is `1010111`. The 3 bits starting at bit location 4 are `101`, which is the binary form of the decimal integer 5, so 5 is the number that is sent out the outlet.
goto	In left inlet: The word goto, followed by a number, sets a pointer to the address specified by the number. The pointer is set at the beginning of the **table** initially.
inv	In left inlet: The word inv, followed by a number, finds the first value which is greater than or equal to that number, and sends the *address* of that value out the left outlet.
length	In left inlet: Sends the length (size) of the **table** out the left outlet.
load	In left inlet: Puts the **table** in load mode. In load mode, every number received in the left inlet gets stored in the **table**, beginning at address 0 and continuing until the **table** is filled (or until the **table** is taken out of load mode by a normal message). If more numbers are received than will fit in the size of the **table**, excess numbers are ignored.
max	Sends the maximum value stored in the **table** out the left outlet.
min	Sends the minimum value stored in the **table** out the left outlet.
next	In left inlet: Sends the value stored in the address pointed at by the goto pointer out the left outlet, then sets the pointer to the next address. If the pointer is currently at the last address in the table, it *wraps around* to the first address.
normal	In left inlet: Undoes a prior load message; takes the **table** out of load mode and reverts it to normal operation.
open	In left inlet: Opens the **table** object's graphic editor window and brings it to the foreground. Double-clicking on the **table** object in a locked patcher has the same effect.
prev	In left inlet: Causes the same output as the word next, but the pointer is then decremented rather than incremented. If the pointer is currently at the first address in the table, it *wraps around* to the last address.
quantile	In left inlet: The word quantile, followed by a number, multiplies the number by the sum of all the numbers in the **table**. This result is then divided by 2^{15} (32,768).

	Then, **table** sends out the address at which the sum of all values up to that address is greater than or equal to the result.
read	In left inlet: The word read, followed by a name, opens and reads data values from a file in Text or Max binary format. Without an argument, read opens a standard Open Document dialog for choosing a file to read values from. If the file contains valid data, the entire contents of the existing table are replaced with the data.
refer	In left inlet: The word refer, followed by the name of another **table**, sets the receiving **table** object to read its data values from the named **table**.
send	The word send, followed by the name of a **receive** object, followed by an address number, sends the value stored at that address to all **receive** objects with that name, without sending the value out the **table** object's outlet.
set	In left inlet: The word set, followed by a list of numbers, stores values in certain addresses. The first number after the word set specifies an address. The next number is the value to be stored in that address, and each number after that is stored in a successive address.
setbits	In left inlet: Changes the value of one or more specific bits of a number stored in the **table**. The word setbits is followed by four number arguments. The first argument is the address being referred to; the second argument is the starting bit location in the number stored at that address (the bit locations are numbered 0 to 31, from the least significant bit to the most significant bit); the third argument specifies how many bits to the right of the starting bit location should be modified, and the fourth argument is the value (stated in decimal or hexadecimal form) to which those bits should be set. For example, the message setbits 47 5 3 6 will look at address 47 in the **table**, start at bit location 5 (the sixth bit from the right), and replace the 3 bits starting at that location with the bits `110` (the binary equivalent of the decimal integer 6). Suppose that address 47 of the table stores the number 87. The binary form of 87 is `1`**`010`**`111`, so replacing the 3 bits starting at bit location 5 with `110` would change the number to `1`**`110`**`111`, which is the binary form of the decimal integer 119. The new number stored at address 47 in the **table** will therefore be 119.
size	In left inlet: The word size, followed by a number, sets the size of the **table** to that number.
sum	In left inlet: Sends the sum of all the values in the **table** out the left outlet.
wclose	In left inlet: Closes the graphic editing window associated with the **table** object.
write	In left inlet: Opens a standard save file dialog for choosing a name to write data values from the table. The file can be saved in Text or Max binary format.

(mouse) The values stored in **table** can be entered and edited graphically with the mouse. When a **table** object is first created in a patcher window, the **table** object's graphic editing window is opened, in which values can be entered by drawing with the mouse. The editing window provides a palette of graphic editing tools.


show crosshairs, to aid precision drawing
select a region to cut, copy, clear, or paste
drag the display, to see another part of the table
draw in values freehand with the mouse
draw values in a straight line, from click to click
1:1
zoom the horizontal display in or out
1:1
zoom the vertical display in or out

When the patcher window is locked, the graphic editing window can be opened by double-clicking with the mouse on the **table** object.

A **table** can be created in a separate file by opening a new Table window and choosing the Save command from the File menu. A **table** can also be created in a separate file by opening a new Text file, and simply beginning the file with the word table. The word table should be followed by a list of space-separated numbers, specifying values to be stored in the **table**.

A **table** which has been saved as a file can be viewed and edited as text by choosing Open as Text… from the File menu. Numbers in the form of text can be pasted in from other sources such as the editing window of a **capture** object, or even from another program such as a word processor. Text from a **capture** object can also be pasted directly into a **table** object's graphic editing window.

Inspector

The behavior of a **table** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **table** object displays the **table** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

Table Size determines the number of values stored in the **table**. A newly created **table** has 128 values, indexed with numbers from 0 to 127.

Table Range determines the range of values which can be displayed on the y axis of the editing window. A newly created **table** has a range of 128, from 0 to 127.

If *Save Table with Patcher* is checked, the values in the **table** are saved as part of the patch that contains it. Otherwise, the **table** has to be saved in a separate file to retain its values.

If *Don't Save* is checked, Max will not ask if you want to save changes made to the **table**, when the patch containing that **table** is closed.

If *Use Note Name Legend* is checked, values are shown on the *y* axis as MIDI note names, rather than numbers.

If *Signed Values* is checked, **table** displays negative numbers as well as positive. In effect, the range of displayed values specified by *Range* is doubled when the *Signed* option is checked, since the range goes in both directions from 0.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

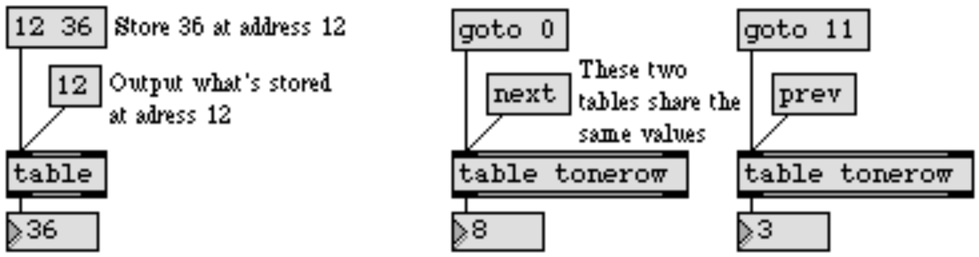
Arguments

any symbol	Optional. The argument gives a name to the **table**. Max looks for a **table** of the same name which has been saved as a separate file. If two or more **table** objects share the same names, they also share the same values, even if Max couldn't find a file with the name.

Output

int	All numbers sent out by **table** are sent out the left outlet.
bang	When the contents of a **table** have been changed by an edit in the graphic editing window, bang is sent out the right outlet.

Examples



An array of any size and range can be stored, recalled, and modified

See Also

capture	Store numbers to view or edit
coll	Store and edit a collection of different messages
funbuff	Store x,y pairs of numbers together
histo	Make a histogram of the numbers received
multislider	Multiple slider and scrolling display
text	Format numbers as a text file
Tutorial 32	The **table** object
Timeline	Creating a graphic score of Max messages
Data Structures	Ways of storing data in Max
Quantile	Using **table** for probability distribution
Tables	Using the **table** graphic editing window

tan

Input

float or int	Input to a tangent function.
bang	In left inlet: Calculates the tangent of the number currently stored. If there is no argument, **tan** initially holds 0.

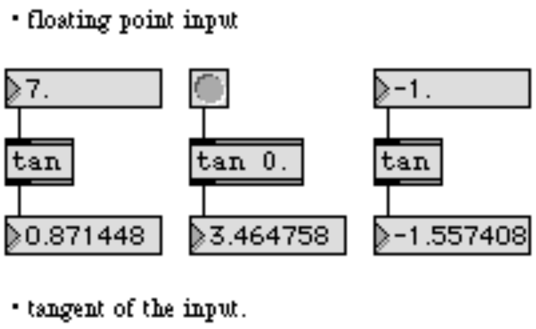
Arguments

float or int	Optional. Sets the initial value for the tangent function.

Output

float or int	The tangent of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tanh	Hyperbolic tangent function

tanh

Input

float or int	Input to a hyperbolic tangent function.
bang	In left inlet: Calculates the hyperbolic tangent of the number currently stored. If there is no argument, **tanh** initially holds 0.

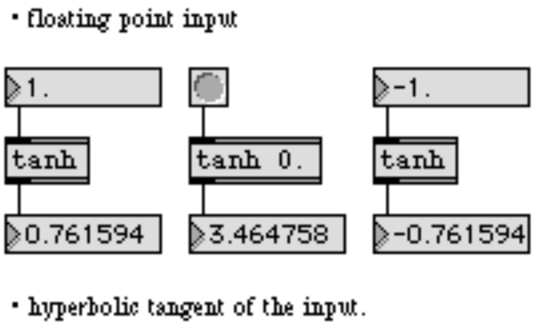
Arguments

float or int	Optional. Sets the initial value for the hyperbolic tangent function.

Output

float or int	The hyperbolic tangent of the input.

Examples



See Also

acos	Arc-cosine function
acosh	Hyperbolic arc-cosine function
asin	Arc-sine function
asinh	Hyperbolic Arc-sine function
atan	Arc-tangent function
atan2	Arc-tangent function (two variables)
atanh	Hyperbolic arc-tangent function
cos	Cosine function
cosh	Hyperbolic cosine function
sin	Sine function
sinh	Hyperbolic sine function
tan	Tangent function

tempo

Output numbers at a metronomic tempo

Input

bang	In left inlet: Starts the **tempo** object's metronome process, or restarts it if **tempo** is already on.
stop	In left inlet: Stops **tempo**.
int	In left inlet: If the number is not 0, it has the same effect as bang. If the number is 0, it has the same effect as stop.
int or float	In 2nd inlet: The number is stored as the tempo, in beats per minute (quarter notes per minute). The tempo is limited between 5 and 300 beats per minute.
	In 3rd inlet: The number is a *beat multiplier,* which can lengthen the amount of time taken for one beat. It slows the tempo down by a factor. For example, a multiplier of 2 will make **tempo** send out its output half as fast.
	In right inlet: The number is the rhythmic value sent out by **tempo**, specified as a fraction of a whole note. For example, the number 8 causes **tempo** to output eighth notes, relative to the specified (quarter note) tempo. The numbers sent out the outlet cycle continuously between 0 and the number 1 less than the rhythmic value. The divisions of a whole note must be between 1 and 96.
tempo	In left inlet: The word tempo, followed by a float, sets the current tempo to the number.
clock	The word clock, followed by the name of an existing **setclock** object, sets **tempo** to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets **tempo** back to using Max's regular millisecond clock.

Arguments

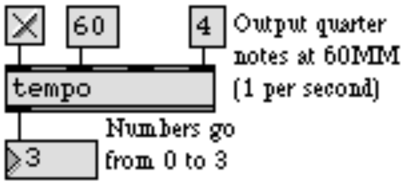
int or float	Optional. The first argument sets an initial tempo, from 5 to 300 beats per minute. If there is no argument, the initial tempo is 120 beats per minute. The second argument is the beat multiplier and is set to 1 by default. The third argument sets an initial rhythmic value of the output, from a whole note (1) to a 64th note triplet (96). If the argument is not present, the initial value is 16.

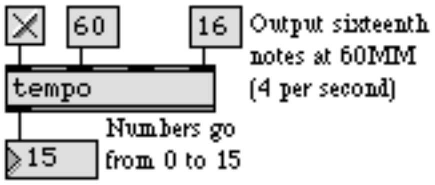
Output

int	When **tempo** is started it outputs numbers in a continuous cycle from 0 to the number 1 less than the specified rhythmic value. The speed at which the numbers are sent out is determined by the tempo (quarter note beats per minute) and the rhythmic value of the output (fraction of a whole note).

tempo

Examples





The tempo (60) defines the speed of a quarter note, division defines the pulse to be sent out

See Also

clocker	Report elapsed time, at regular intervals
metro	Output a bang message at regular intervals
setclock	Control the clock speed of timing objects remotely
Tutorial 31	Using timers

text

Input

clear	Erases the contents of **text**.
cr	Puts a carriage return at the end of the contents of **text**, to start a new line. If the last character in **text** is a space, the carriage return replaces that space.
line	The word line, followed by a number, causes **text** to send out the contents of that line number (up to 256 characters) with the word set prepended (for setting the contents of a **message** box). Lines are numbered beginning with 1; any line number message less than 1 is converted to line 1. If a nonexistent line number is requested, nothing is sent out.
open	Opens the object's text window for editing. Double-clicking on the **text** object in a locked patcher has the same effect. The **text** object ignores messages to change its text while the editing window is open. Unlike the **capture** object, changes made in the editing window of **text** actually alter the contents of the object.
read	The word read, followed by a symbol that specifies a filename, will read the contents of a text file of up to 32,000 characters into the **text** object. If no filename or pathname is specified, the read message will call up the standard Open Document dialog box, so that a text file can be specified.
settitle	The word settitle, followed by any word, sets the title of the text window. If you want more than one word to appear as the default text, you must enclose the words in double quotes or precede the spaces with a backslash (\).
symbol	The word symbol, followed by any word, stores that word at the end of the contents of **text**. This is useful if you want to store a word that would otherwise be understood as a specific message by **text**. For example, symbol clear stores the word clear, followed by a space, at the end of the contents of **text**, rather than erasing the contents.
tab	Puts a tab stop at the end of the contents of **text**. If the last character in **text** is a space, the tab stop replaces that space.
wclose	Closes the window associated with the **text** object.
write	The word write, followed by a symbol that specifies a filename, will save the contents of **text** as a text file in the current default folder unless the file is specified with an absolute pathname. If no filename or pathname is specified, the write message will open up a standard Save As dialog box, so that the contents of **text** can be saved in a separate text file.
anything else	The message is stored in the **text** object, placed after any previously stored messages, and is followed by a space.

text

(mouse)	Double-clicking with the mouse on the **text** object (when the patcher window is locked) opens an editing window in which the contents of **text** can be viewed and edited. The **text** object ignores messages to change its text while the editing window is open. Unlike the **capture** object, changes made in the editing window of **text** actually alter the contents of the object.

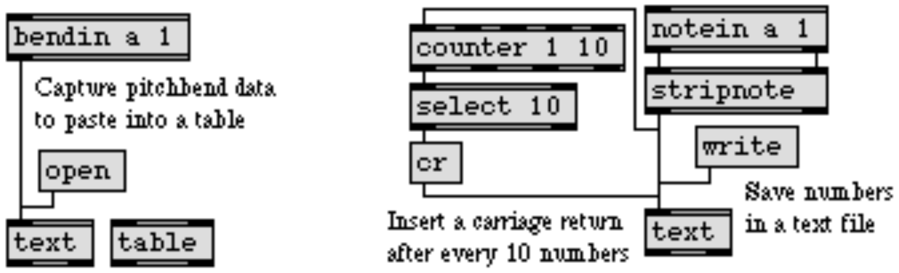
Arguments

symbol	Names a text file to be read in when the object is loaded.

Output

set	When a line message is received, the text of the specified line number is sent out preceded by the word set. The message can be used to set the contents of a **message** box (or can be sent to any other object for which that particular set message is appropriate).

Examples


bendin a 1
Capture pitchbend data to paste into a table
open
text
table
counter 1 10
select 10
cr
Insert a carriage return after every 10 numbers
notein a 1
stripnote
write
Save numbers in a text file
text

Collect messages as text, to paste elsewhere or to save as a separate file

See Also

capture	Store numbers to view or edit
filein	Read in a file of binary data
spell	Convert input to ASCII codes
sprintf	Format a message of words and numbers
table	Store and graphically edit an array of numbers
textedit	Object for user-entered text in a patcher

textedit

User-entered text in a patcher

Input

(typing)	When the **textedit** object is highlighted, typing enters text into the text display area and modifies its buffer, unless the object is set to read-only mode (see the readonly message). The ASCII value of the character typed is sent out the middle outlet.
(mouse)	Clicking with the mouse on the **textedit** object (when the patcher window is locked) will cause the **textedit** object to send either the letter or word selected out its right outlet depending on the setting of the click mode (see the clickmode message).
bang	Outputs the typed or stored contents of the **textedit** object's buffer.
append	The word append, followed by a message, will append the message to the **textedit** object's buffer without causing any output.
autoscroll	The word autoscroll, followed by a 0 or 1, toggles autoscrolling in the text display area. The message autoscroll 1 lets you scroll past the amount of text displayed in the **textedit** window when the number of lines is set to 1 and the word wrapping is disabled (see the wordwrap message) using either the cursor or by clicking and dragging in the **textedit** window. The default is 0 (autoscroll disabled).
brgb	The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **textedit** object. The default value is white (brgb 255 255 255).
clear	Erases the contents of the **textedit** object's buffer.
clickmode	The word clickmode, followed by a 0 or 1, sets the way that the **textedit** object responds to mouse clicks in the text display area. The message clickmode 0 will send an individual *character* clicked on out the right outlet of the **textedit** object. Setting the object with the message clickmode 1 will send the *word* the user clicks on. The default is 0 (select characters).
frgb	The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the text displayed by the **textedit** object. The default value is black (frgb 0 0 0).
keymode	The word keymode, followed by a 0 or 1, sets the way that the **textedit** object responds to carriage returns while typing characters into its text display area. The message clickmode 0 allows for text input, and displays carriage returns normally. Setting the object with the message keymode 1 causes the carriage return to output the entire contents of the current buffer. The default is 0.
lines	The word lines, followed by a number, sets the maximum number of lines of text that **textedit** will display. lines 0 removes any limit on the number of text lines. You'd want to use lines 1 on a **textedit** object that is being used to enter a number or word in a "dialog box" context. The default is that there is no line limit.

outputmode	The word outputmode, followed by a 0 or 1, sets whether the **textedit** object outputs its contents as a message or as a single symbol. The message outputmode 0 causes the output of the object to be sent out as messages. Setting the object with the message outputmode 1 will output the buffer contents as a single symbol. The default is 0 (output as messages).
readonly	The word readonly, followed by a 0 or 1, toggles the read only mode of the **textedit** object. The message readonly 1 disables any user entry into the text box. Messages which operate on the current contents of the **textedit** buffer such as clear, append, or separator are not affected by the readonly message. The default is 0 (readonly mode off).
set	The word set, followed by any message, sets the contents of the **textedit** object's buffer while causing no output.
select	Causes the text (if any) to be highlighted, and if the object is not in read-only mode, sets the object to be the target of keyboard events.
separator	The word separator, followed by any symbol, sets that symbol as a line separator. and treats it as a carriage return when the contents of the buffer are output. If the buffer contains the text "red green blue" and the object receives the message separator green, the next bang received by **textedit** will output red *(carriage return)* blue.
wordwrap	The word wordwrap, followed by a 0 or 1, sets the way that the **textedit** object displays messages which are longer than the **textedit** display area. The message wordwrap 0 (default) will enable text wrapping on word boundaries in the display area. The message clickmode 1 disables word-wrap.
(Font menu)	The size and font used in the **textedit** object can be altered by choosing a different font or size from the Font menu.

Inspector

The behavior of a **textedit** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **textedit** object displays the **textedit** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

Typing numbers into the *Maximum Lines* number box sets the maximum lines displayed in the text area. The default is 0. *Options* contains three checkboxes which set the behavior and output of the **textedit** object. By default, none of these options are selected. Checking *Read-only* sets the object to display text only. Checking the *Return Enters Text* checkbox causes the carriage return to output the entire contents of the current buffer on a carriage return. If *Output as One Symbol* is checked, the **textedit** object will output its contents as a single symbol rather than as a message. Text wrapping on word boundaries can be enabled by checking the *Word Wraparound* option, and the *Automatic Scrolling* option (default on) allows the scrolling of selected text. The output behavior of the **textedit** object is also set using the *When Clicked....* checkboxes. You can choose to output characters (the default) or words when you click on the text.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

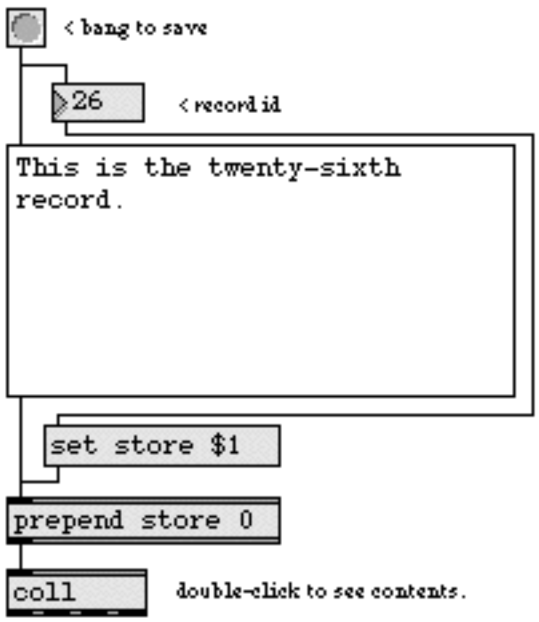
Arguments

None.

Output

symbol	Out left outlet: The currently stored contents of the **textedit** object's buffer are output when the object receives a bang message. If the **textedit** message has been set to enter text on a carriage return using the keymode 1 message, a carriage return will also output the typed text and the buffer contents.
symbol	Out middle outlet: The ASCII value of the typed key.
symbol	Out right outlet: The word or letter in the **textedit** object's text box that the user has clicked on.

Examples


< bang to save
26
< record id
This is the twenty-sixth record.
set store $1
prepend store 0
coll
double-click to see contents.

Collect text to store in a **coll** *object*

See Also

dialog	Open a dialog box for text entry
text	Format messages as a text file

thispatcher

Send messages to a patcher

The **thispatcher** object is placed *inside* the patcher you want to control. It sends messages to the patcher that contains it.

Input

loadbang	Sending the loadbang message to **thispatcher** causes any **loadbang** objects in the same patcher to send out a bang. Any other objects which use a loadbang message internally for initialization (such as the **preset** object) will receive this message, too.
front	Brings the patcher window to the front, or opens the window and brings it to the front if it's loaded as a subpatch but is not visible.
wclose	Closes the patcher window. If the patcher has been edited, you will be asked if you want to save the changes.
clean	Resets the patcher window's "dirty" flag, so the user won't be asked to save changes when the window is closed.
dirty	Sets the patcher window's "dirty" flag, so the user will be asked to save changes when the window is closed.
dispose	Permanently closes the patcher window and frees its memory. You can use this in conjunction with the load message to the **pcontrol** object to open and close patchers automatically. If the patcher has been edited, you will be asked if you want to save the changes.
offset	For patchers contained inside boxes (using the **bpatcher** object), the offset message sets the upper left corner of the visible portion of the patcher in the box. The word offset should be followed by two numbers; the first number specifies the left offset (in pixels) and the second specifies the top offset. By default, patchers in **bpatcher** boxes are displayed with an offset of 0,0. When you hold down the Command and Shift keys on Macintosh or the Control and Shift keys on Windows and drag in a **bpatcher** object's box, the offset changes as you move, and the current offset is displayed in the Assistance area of the window that contains the **bpatcher**. You can use these numbers to help you determine appropriate arguments to the offset message.
path	If the patcher window is saved as a file, the word path sends the full pathname of folder containing the patcher's file out the **thispatcher** object's right outlet.
write	Saves the patcher's file if it has a name; otherwise, brings up a Save As dialog.
(others)	**thispatcher** will respond to messages to create new objects. The format of these messages is cryptic and subject to change, but you can get some idea of what might be worth trying by examining a patcher file as text, and trying any of the

messages that begin with #P. Leave the #P out of the message you send to **thispatcher**. Use of **thispatcher** to create new objects is not supported.

patcher The word patcher, followed by any text, replaces the window name shown in the title bar. The new window name is shown enclosed in brackets, to indicate that it is not the actual file name, which is left unaltered.

window window notitle hides the title bar of the patcher window. window title shows the title bar. window flags noclose hides the close box that normally appears in the title bar of the patcher window. window flags close shows the close box. window flags nozoom hides the zoom box that normally appears in the right corner of the title bar. window flags zoom shows the zoom box. window flags nogrow hides the scroll bars and the grow box that normally appears in the lower right portion of the window. window flags grow shows the scroll bars and the grow box. window size, followed by four numbers, sets the precise screen coordinates (in pixels from the top left corner of the screen) of the left, top, right, and bottom limits of the window, respectively. The left and top coordinates refer to the upper left corner of the content portion of the window, not the title bar.

window fullscreen 1 hides the menu bar and resizes the patcher window to fill the entire screen, with no title bar and no scroll bars. window fullscreen 0 shows the menu bar and restores the previous size and appearance of the patcher window.

The above window messages do not take effect until you send the message window exec.

The messages window getsize, window getflags, and window gettitle, cause **thispatcher** to send a window message out the left outlet reporting the current characteristics of the window.

savewindow The word savewindow, followed by a non-zero number, means that any unusual window settings caused by window flags messages to **thispatcher** will be saved as part of the patch the next time the patch is saved. The message savewindow 0 means that changes to the window caused by window flags messages to **thispatcher** will not be retained when the patch is saved; the prior patcher window settings are saved. If no savewindow message has been received, the patcher will be saved with a normal window appearance.

Scripting Messages

The script message to **thispatcher** permits dynamic control over object creation, deletion, sizing and positioning, and patching. The word script is followed by a *keyword* that indicates a function. Following the keyword are *arguments* that specify what objects are to be affected by the message.

In the discussion of each script message that follows, the syntax indicates required arguments for the message after the keyword in angle brackets. An example of each message is also provided.

A *variable-name* is a symbol that names either a new or existing object. You can set variable names by choosing **Name...** from the Object menu, or with certain scripting messages such as new and select.

thispatcher

Send messages to a patcher

Instantiating and Deleting Objects

new
: Creates a new object in a patcher window and gives it a name.

Syntax: script new <variable-name> <creation message>

Example: script new footog toggle 101 93 15 0

Creates a new **toggle** object 15 pixels square at 101 93 and assign it to the variable footog.

Since the save formats of Max objects are not documented, in order to determine the appropriate creation message for the desired object, you'll have to examine Max patchers as text. Most objects are saved with one of the following basic styles:

#P classname arguments; (internal UI object)

#P newex classname arguments; (normal internal or external object)

#P user classname arguments; (external UI object)

Remove the #P and the semicolon and put the rest of the message after the variable name that will be assigned to the new object.

delete
: Deletes an object in a patcher window.

Syntax: script delete <variable-name>

Example: script delete footog

Deletes the object associated with the variable name footog.

hidden
: Specifies that an object (or connection) will be hidden when created.

Example: script hidden new footog toggle 101 93 15 0

Creates a hidden object associated with the variable name footog. The hidden keyword can also be used when specifying connections between objects.

Assigning Variable Names to Objects

class
: Assigns a variable name to the first instance of a specified class with matching arguments

Syntax: script class <variable-name> <class-name> <arguments (optional)>

Example: script class rubadub + 4

Assigns the name rubadub to the first instance found of **+** with argument 4 in the patcher.

nth Assigns a variable name to the *nth* instance of a specified class

Syntax: script nth <variable-name> <class-name> <index>

Example:script nth yoyo toggle 1

Assigns the name yoyo to the first **toggle** found in the patcher.

The order of objects in a patcher is determined by the front-to-back ordering. Objects in back of the patcher that draw behind other objects are first in the search order.

selected Assigns a variable name to the first object found that is selected

Syntax:script selected <variable-name>

Example:script selected impo

Assigns the name impo to the first object found that is selected. Obviously this script message only works when the patcher is unlocked, since no object can be selected in a locked patcher.

Connecting and Disconnecting Objects

For all three connection messages described below, inlets and outlets are specified by index, with 0 denoting the leftmost inlet or outlet. The first variable specified is the object whose outlet you are connecting or disconnecting and the second variable is the one whose inlet you are connecting. Messages can then flow from outlet to inlet.

connect Connects two objects together with a patch cord

Syntax:script connect <outlet-variable-name> <outlet-index> <inlet-variable-name> <inlet-index>

Example:script connect fooboo 0 bobo 0

Connects the left outlet of the object with the variable name fooboo to the left inlet of the object with the variable name bobo.

Note: Adding the keyword hidden (e.g., script hidden connect fooboo 0 bobo 0) creates hidden connections.

disconnect Disconnect two objects connected by a patch cord

Syntax:script disconnect <outlet-variable-name> <outlet-index> <inlet-variable-name> <inlet-index>

Example:script disconnect fooboo 0 bobo 0

This message undoes the connection between the left outlet of fooboo and the left inlet of bobo.

connectcolor Modify the color of an existing patch cord, setting it to one of Max's 16 standard colors.

Syntax: script connectcolor <outlet-variable-name> <outlet-index> <inlet-variable-name> <inlet-index> <color>

Example: script connectcolor rover 0 dover 2 12

Changes the color of the connection between the left outlet of the rover object with the 3rd inlet of the dover object to the color stored at index 12.

Changing Object Properties

hide Hide a visible object.

Syntax: script hide <variable-name>

Example: script hide visigoth

Hides the object named visigoth

show Show a hidden object.

Syntax: script show <variable-name>

Example: script show visigoth

Makes the object named visigoth visible.

ignoreclick Set an object not to respond to mouse clicks.

Syntax: script ignoreclick <variable-name>

Example: script ignoreclick visigoth

Makes the object named visigoth ignore mouse clicks.

respondtoclick Set an object to respond to mouse clicks.

Syntax: script respondtoclick <variable-name>

Example: script respondtoclick visigoth

Makes the object named visigoth respond to mouse clicks.

bringtofront Bring an object to the front of the layer it's currently in.

Syntax: script bringtofront <variable-name>

Example: script bringtofront visigoth

If visigoth is in the foreground layer, this message moves it to the front of the foreground layer. Otherwise it moves it to the front of the background layer.

sendtoback Move an object to the back of the layer it's currently in.

Syntax: script sendtoback <variable-name>

Example: script sendtoback visigoth

If visigoth is in the foreground layer, this message moves it to the back of the foreground layer. Otherwise it moves it to the back of the background layer. Note that objects that are "in the back" are the first objects to be found by the variable assignment messages nth and class.

size Change an object's size. There are some objects that have restrictions on their size, but they generally do not protect themselves against sizes they don't expect, so use this message with some caution. For instance the **toggle** object expects to be a square. It may not draw properly if it's made into a rectangle.

Syntax: script size <variable-name> <width> <height>

Example: script size togipoo 30 30

Changes the object named togipoo to be 30 by 30 pixels.

Sending Messages to Objects

send Send a message to an object. This message is the same as using a **message** box with a semicolon or a **send** object, but you use the object variable name feature of scripting to specify the object that will receive the message—using script send to communicate with a named **receive** object does not work. The message can only be sent to an object within the patcher as the **thispatcher** object receiving the script send message.

Syntax: script send <variable-name> <message>

Example: script send foobert 666

The object with the variable name foobert receives an int 666 message. If foobert were a number box, its displayed value would change to 666.

sendbox Send a message to an object box. This message is identical to **send** except that it sends the message to an object's box rather than the object referred to by the box. There is currently only one object, **bpatcher**, in which the object and box are different objects. The box is a **bpatcher**, and the object is a patcher. What can you tell a **bpatcher** to do? One example is the boxborder message, which is equivalent to sending the border message to a **thispatcher** object in a patcher inside a **bpatcher**. Peek inside the Inspector patch for **bpatcher** for other ideas.

Syntax: script sendbox <variable-name> <message>

Example: script sendbox bpbp boxborder 0

If bpbp names a **bpatcher** object, this script message would tell it not to draw its border.

Moving Objects

move — Move an object to an absolute position relative to the current top-left corner of a patcher window. Note that the 0,0 point is underneath the icon bar.

Syntax: script move <variable-name> <top> <left>

Example: script move molly 0 100

Moves the object named molly to the left edge of the window, 100 pixels down from the top.

offset — Move an object a distance from its current position. Positive distances move the object down and to the right, negative distances move it up and to the left.

Syntax: script offset <variable-name> <delta-x> <delta-y>

Example: script offset molly 30 -40

Moves the object named molly 30 pixels to the right and 40 pixels up.

offsetfrom — Move an object a set distance from another object.

Syntax: script offsetfrom <variable-name-to-move> <target-variable-name> <delta-x> <delta-y> <top-left-flag>

The top-left-flag is 1 if the distance is relative to the top-left corner of the object, and 0 if it is relative to the bottom-right corner.

Example: script offsetfrom molly panther -100 -120 1

Moves the object named molly 100 pixels to the left of the left side of the object named panther, and 120 pixels above the top of the object named panther.

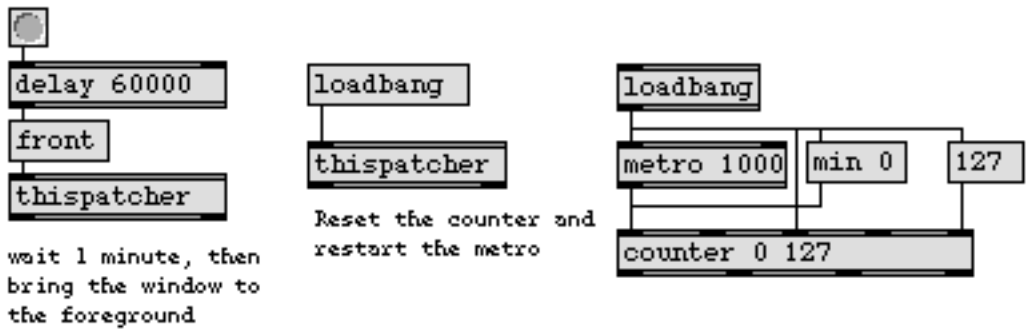
Arguments

None.

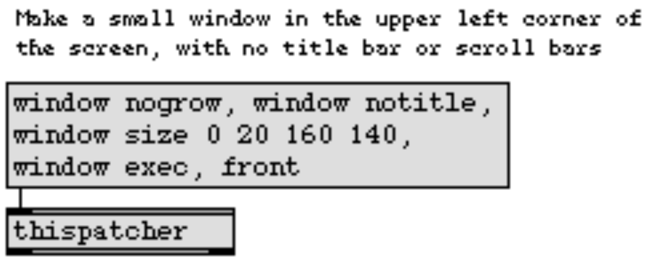
Output

window	Out left outlet: When the message window getsize is received, **thispatcher** sends out the words window size followed by the screen coordinates (in pixels from the top left corner of the screen) of the left, top, right, and bottom limits of the window. When the message window gettitle is received, the message window title or window notitle is sent out, depending on whether the window has a title bar. When the message window getflags is received, **thispatcher** sends out the words window flags followed by the visibility of the scroll bars and grow box (grow or nogrow), the close box (close or noclose), and the zoom box (zoom or nozoom).
symbol	Out right outlet: The full pathname of the folder or volume containing the patcher's file in response to the path message. If the patcher has not been saved, there is no output.

Examples


delay 60000
front
thispatcher
wait 1 minute, then bring the window to the foreground
loadbang
thispatcher
Reset the counter and restart the metro
loadbang
metro 1000
min 0
127
counter 0 127

Automatic window control, file saving, or patcher reset are possible with **thispatcher**


Make a small window in the upper left corner of the screen, with no title bar or scroll bars
window nogrow, window notitle, window size 0 20 160 140, window exec, front
thispatcher

Windows can have any size, location, and appearance, set within the patch itself

See Also

bpatcher	Embed a visible subpatch inside a box
pack	Combine numbers and symbols into a list
patcher	Create a subpatch within a patch
pcontrol	Open and close subwindows within a patcher
pvar	Connect to a named object in a patcher
sprintf	Format a message of words and numbers
Tutorial 46	Basic Scripting
Tutorial 47	Advanced Scripting

thistimeline

Send messages to a timeline

Input

any message	If **thistimeline** is in an action patch, and the action is currently being used in a timeline, then any message that would normally be acceptable to a **timeline** object can be received by **thistimeline**, and will be transmitted to the timeline that contains the action.
bang	Sends out the current time of the timeline that contains the **thistimeline** object in an action.

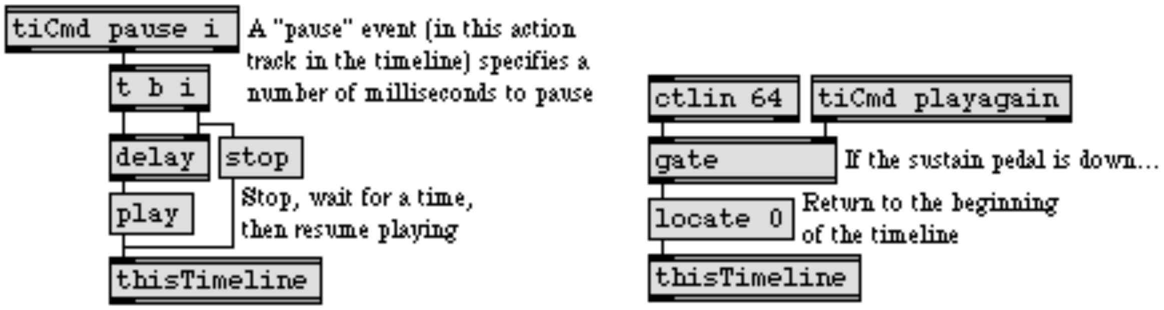
Arguments

None.

Output

(to timeline)	The messages received in the inlet are conveyed to the timeline that contains the action in which the **thistimeline** object is located.
int	When bang is received in the inlet, **thistimeline** sends out its outlet the current time, in milliseconds, of the timeline that contains it in an action.

Examples


tiCmd pause i
A "pause" event (in this action track in the timeline) specifies a number of milliseconds to pause
t b i
delay
stop
play
Stop, wait for a time, then resume playing
thisTimeline
ctlin 64
tiCmd playagain
gate
If the sustain pedal is down...
locate 0
Return to the beginning of the timeline
thisTimeline

A **timeline** *can actually control itself via a* **thistimeline** *object in an action*

See Also

thistrack	Send messages to a **timeline** track
ticmd	Receive messages from a **timeline**
timeline	Time-based score of Max messages
Tutorial 41	Timeline of Max messages
Timeline	Creating a graphic score of Max messages

thistrack

Send messages to a timeline track

Input

any message	If **thistrack** is in an *action* patch, and the action is currently being used in a **timeline**, then a message received by **thistrack** will be transmitted to the timeline track that is calling the action.
mute	The word mute, followed by a nonzero number, mutes the timeline track of the action that contains **thistrack**. The message mute 0 unmutes the track.
name	The word name, followed by any other symbol, sets the name of the action's timeline track (in the graphic timeline editor window) to that symbol.
height	The word height, followed by a number greater than 0, sets the height, in pixels, of the timeline track's visual display in the graphic timeline editor window.

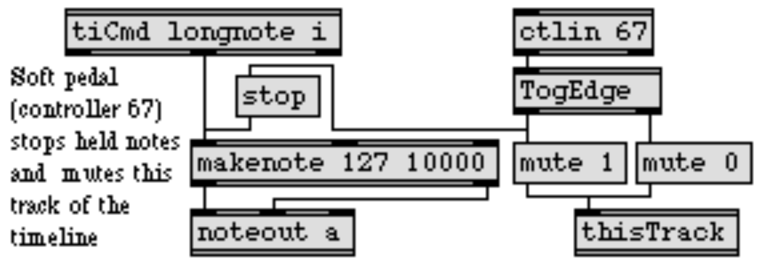
Arguments

None.

Output

(to timeline)	The messages received in the inlet are applied to the timeline track that is using the action containing the **thistrack** object.

Examples



A **timeline** *action can mute its own track with a* **thistrack** *object*

See Also

thistimeline	Send messages to a **timeline**
ticmd	Receive messages from a **timeline**
timeline	Time-based score of Max messages
Timeline	Creating a graphic score of Max messages
Tutorial 41	Timeline of Max messages

thresh

Combine numbers into a list, when received close together

Input

int or float	a time In left inlet: Numbers are combined into a list if received within a certain time of each other. When the time between incoming numbers is greater than the specified threshold, the list is sent out the outlet, and a new list is started. In right inlet: The number is stored as the time, in milliseconds, to wait before sending out the compiled list of numbers. If no new number is received in the left inlet within that time, the list is sent out and a new list is started.
list	In left inlet: The entire list is appended to the list stored in **thresh**.

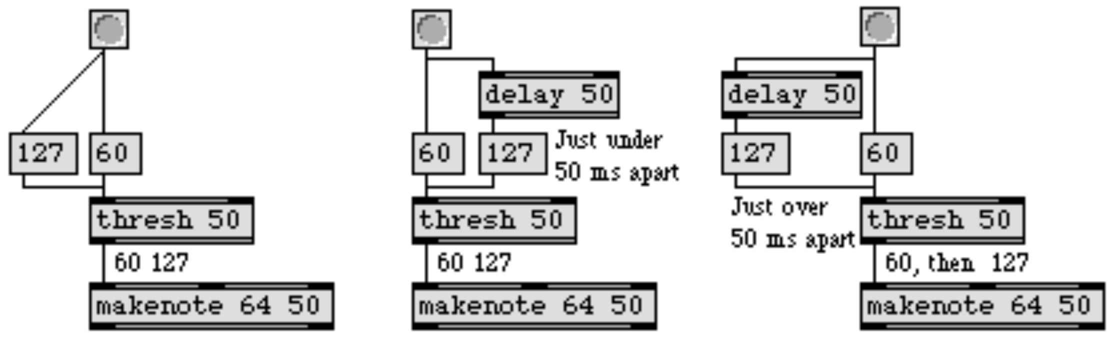
Arguments

int	Optional. Sets an initial value for the threshold time. If no argument is present, the initial value is 10 milliseconds.
float	Converted to int.

Output

list	Each number received in the left inlet is appended to a list stored by **thresh**. If a certain time passes without a new number being received, **thresh** sends out the list and starts a new list.

Examples



If threshold time is exceeded without a new number being received, **thresh** *sends out what it holds*

See Also

bondo	Synchronize a group of messages
buddy	Synchronize arriving data, output them together
iter	Break a list up into a series of numbers
pack	Combine numbers and symbols into a list
zl	Multi-purpose list processor
Tutorial 37	Data structures

Input

The **ticmd** object is intended to be placed in an action patch, which is loaded as a track in a timeline. **ticmd** gets its input from an *event editor* of the same name in the timeline track. The type(s) of message(s) it can receive depends on the typed-in argument(s) i, f, l, b, s, or a.

int If the second (and last) typed-in argument is i, then **ticmd** receives an int value from a timeline event editor, and passes the number out its middle outlet. There are three types of event editor that can be placed in a track of a timeline for sending int values: **int**, **etable**, and **efunc**.

The **int** event editor in a timeline looks like, and functions much like, a **number box** object in a patcher. When the timeline is being played and reaches the **int** event editor, it sends the value in the **number box** to the appropriate **ticmd** object, to be passed out the **ticmd** object's middle outlet.

The **etable** event editor is similar to the **table** object. It is an array of ints which can be edited graphically. When the timeline is being played and it reaches an **etable** event editor, it sends out all the numbers in the **etable** one-by-one at a rate proportional to the space that the **etable** occupies on the timeline. For example if an **etable** containing 128 values occupies the space from time 1000 to time 9000 (in milliseconds) on a timeline track, then **ticmd** will receive ints at the rate of 16 per second as the timeline progresses through those eight seconds.

The **efunc** event editor is a two-dimensional array containing pairs of *x*,*y* values which can be edited graphically. When the timeline is being played and it reaches an **efunc** event editor, it sends the *y* values in the **efunc** to **ticmd** at a time determined by the *x* value (relative to the maximum range of *x* values), proportional to the space that the **etable** occupies on the timeline. For example, if the maximum range of x values in an **efunc** is 1000, and the **efunc** covers a time period from 1000 to 9000 (in milliseconds), then the *x*,*y* pair 500, 127 would cause the number 127 (the *y* value) to be sent to **ticmd** at time 5000 ($^{500}/_{1000}$ of the way from 1000 to 9000).

float If the second (and last) typed-in argument is f, then **ticmd** receives a float value from a timeline event editor, and passes the number out its middle outlet. The **float** event editor looks and functions like a float **number box** in a patcher window.

list If the second argument is l, or if there are more than two arguments, then **ticmd** receives a list from a **messenger** event editor in the timeline. A **messenger** looks just like a **message box** object except that the name of the event (the name of the **ticmd** object it will send to) is printed at the beginning of the box. (The name will not be sent out as part of the message, however. It's just there to remind you where the message will be sent.)

bang — If the second (and last) argument is b, then **ticmd** receives a bang message from a **messenger** in the timeline, regardless of what message is typed into the **messenger**.

symbol — If the second (and last) argument is s, then **ticmd** receives a symbol from a **messenger** in the timeline. If more than one word is typed into the **messenger**, only the first word gets sent to **ticmd**. To include more than one word in a messenger, and have them all sent out as a single symbol to **ticmd**, precede the space character(s) with a backslash (\).

any message — If the second (and last) argument is a, then **ticmd** can receive any message from a **messenger** in the timeline, and will send it out the middle outlet unchanged.

Arguments

symbol — Obligatory. The first argument is the name of the **ticmd** object, which will appear as a possible event in a timeline track that uses the action containing the **ticmd**. More than one **ticmd** in an action may have the same name, and each one will receive the same message from the timeline event, although the order in which they will receive the message is undefined. **ticmd** objects in the same action with the same name can even have different type arguments (can expect different types of message), but the event editor that appears in the timeline will depend on the type argument of the **ticmd** object that is loaded first (which cannot always be reliably predicted).

i, f, s, l, b, or a — Optional. After the first argument, each additional argument creates a new outlet (in addition to the left and right outlets, which always exist) and specifies the type of message to be sent out of that outlet: i for int, f for float, l for list, b for bang, s for symbol, and a for any message. If there is no type argument present, no middle outlet will be created; the event can still be placed in the timeline track, however, as a **messenger**, and **ticmd** will still send a bang message out its left and right outlets.

If the only type argument is f, the event editor in the timeline track will be a float **number box**. If the only type argument is i, the event editor in the timeline track can be a **number box**, an **etable**, or an **efunc**. (See input message int, above.) If the type argument is anything else, or if there is more than one type argument, the event editor in the timeline track will be a **messenger**. (See input message list, above.)

Output

bang — Out left outlet: When an event with the same name as the **ticmd** is reached in a timeline, a bang is sent out **ticmd** object's left outlet.

Out middle outlet(s): If the outlet has been specified as a b outlet, bang is sent out when the event is reached in the timeline (immediately after the left outlet sends its bang). The word bang sent out of an s outlet has the same effect.

Out right outlet: When the timeline reaches the end of a **messenger** event with the same name as the **ticmd**, a bang is sent out **ticmd** object's right outlet.

int Out middle outlet(s): If the outlet has been specified as an i outlet, an int is sent out when the event is reached in the timeline (immediately after the left outlet sends its bang). A symbol that is actually an integer number (sent out of an s outlet) has the same effect.

float Out middle outlet(s): If the outlet has been specified as an f outlet, a float is sent out when the event is reached in the timeline (immediately after the left outlet sends its bang). A symbol that is actually a decimal number (sent out of an s outlet) has the same effect.

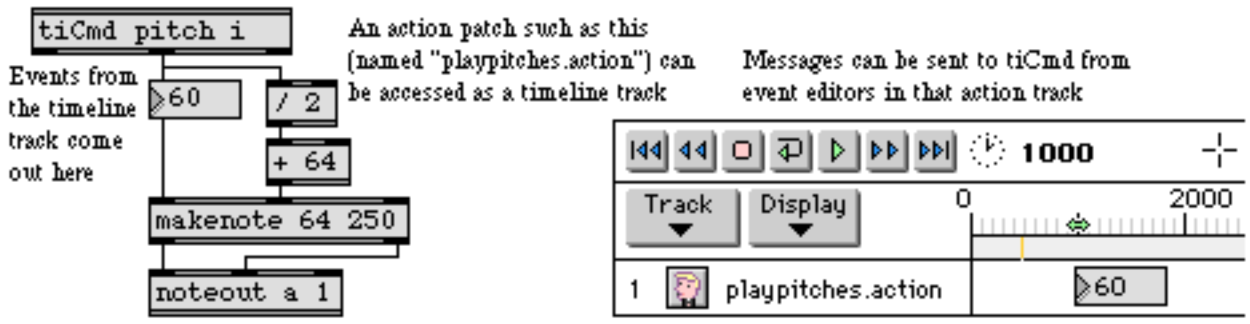
list Out middle outlet(s): If the outlet has been specified as an l outlet, a list is sent out when the **messenger** event is reached in the timeline (immediately after the left outlet sends its bang).

If there are more than two arguments (two or more in addition to the *name* argument) then a list received from the timeline will be broken up and each item in the list will be sent out a different middle outlet, in order from left to right.

symbol Out middle outlet(s): If the outlet has been specified as an s outlet, a symbol is sent out when the **messenger** event is reached in the timeline (immediately after the left outlet sends its bang). However, if the symbol to be sent out the outlet is a number or is bang, then it is sent out as an int, a float, or a bang.

any message Out middle outlet: If the outlet has been specified as an a outlet, the message is sent out when the **messenger** event is reached in the timeline (immediately after the left outlet sends its bang).

Examples


tiCmd pitch i
Events from the timeline track come out here
60
/ 2
+ 64
makenote 64 250
noteout a 1
An action patch such as this (named "playpitches.action") can be accessed as a timeline track
Messages can be sent to tiCmd from event editors in that action track
1000
Track
Display
0
2000
1
playpitches.action
60

A timeline communicates with an action patch via the **ticmd** *object*

See Also

thistimeline	Send messages to a **timeline**
timeline	Time-based score of Max messages
Tutorial 41	Timeline of Max messages
Timeline	Creating a graphic score of Max messages

timeline

Time-based score of Max messages

Input

clock	The word clock, followed by the name of an existing **setclock** object, sets the timeline to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets the **timeline** object back to using Max's regular millisecond clock.
locate	The word locate, followed by a number, specifies a time on the timeline—in milliseconds—and moves the **timeline** object's current time pointer to that time. If the timeline is already playing when a locate message is received, it will continue playing after relocating its current time pointer.
markers	The word markers, followed by an outlet number, causes the first word of each marker event in the timeline to be sent out the specified outlet, as the argument to an append message to be sent to a **umenu** object. (If the specified outlet does not exist, an error message is printed in the Max window and nothing is sent out of the **timeline** object.) Because the markers message is intended for storing the beginning of each marker in a **umenu** object, it first causes the message clear to be sent out the outlet to clear the **umenu** object's previous contents. Immediately after that, a series of append messages is sent out, to add the first word of each marker to the **umenu**. (The text output of the **umenu** object can then be attached to a **prepend** search object, which is in turn **umenu** back to the inlet of the **timeline** object, to locate the current time pointer at a marker location. See the example.)
mute	The word mute, followed by the number of a timeline track, mutes that track, preventing its events from being sent to the action patch.
open	Causes the window associated with the **timeline** object to become visible. The window is also brought to the front. Double-clicking on the **timeline** object in a locked patcher has the same effect.
play	Plays the timeline contained in the **timeline** object.
read	The word read, followed by the name of a timeline file, loads that file into the **timeline** object. The word read by itself calls up a standard Open Document dialog box, so that a timeline file can be read in.
search	The word search, followed by a symbol, searches in the timeline for a marker event in which the first word is an exact match of that symbol. If an exact match is found, the current time pointer of the timeline moves to the location of the matching marker.
stop	Stops the timeline.
timeFormat	The word timeFormat, followed by an integer from 0 to 4, sets the way in which time is displayed in the graphic timeline editor window. The number 0 means *milliseconds*, 1 means *MIDI Clock*, 2 means *24 fps* (frames per second), 3 means *25 fps*, and 4 means *30 fps*. Any other number will be limited to within the 0 to 4 range.

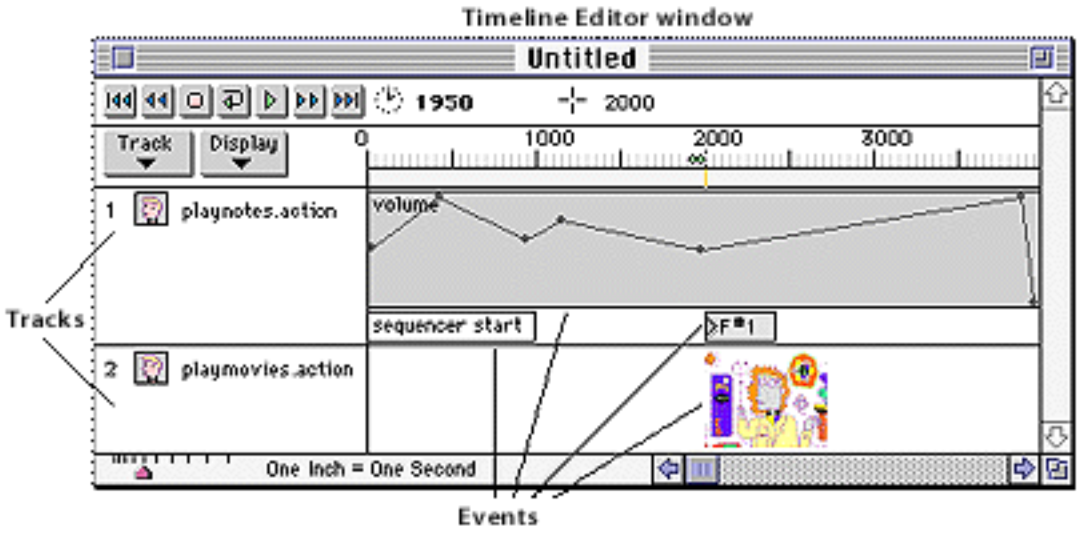
unmute — The word unmute, followed by the number of a timeline track, unmutes the track, allowing its events once again to be sent to the action patch.

wclose — Closes the window associated with the **timeline** object.

write — Calls up the standard Save As dialog box, so that the contents of **timeline** can be saved in a separate file.

zoomLevel — The word zoomLevel, followed by an integer from 0 to 10, will set the magnification of the view of the timeline displayed in the graphic editor window. 0 means maximum zoom out (1 inch = 40 seconds) and 10 means maximum zoom in (1 inch =.04 seconds). The default zoom level of the timeline window is 4 (1 inch = 4 seconds). Any number that exceeds the 0 to 10 range will be limited to stay within the range.

When a **timeline** object is created, it opens a *timeline editor window*, a time-based graphical score of Max messages. Other patches can be loaded into this timeline as individual *tracks* (analogous to tracks of a multi-track sequencer, or staves of a musical score), and messages can be placed in the tracks to be sent to those patches at specific times. A patch that is loaded into a timeline track should generally contain at least one **ticmd** object, to receive messages from the timeline. Such a patch is known as an *action*. The messages in the tracks are known as events, and are entered by placing special *event editor* objects in the tracks.



When the timeline is played, the events in the tracks are sent to specific **ticmd** objects in the action patch, and the event's message goes out the **ticmd** object's outlet.

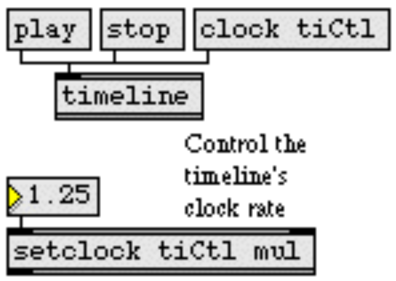
Arguments

symbol	Optional. The first argument specifies the name of a timeline file to read into the **timeline** object. If no file of that name is found, the name will still appear in the title bar of the empty timeline editing window that is opened when the **timeline** object is created.
int	The second argument (or the only argument, if no name argument is present) sets the number of outlets the **timeline** object will have. Any number less than 1 will be set to 0.

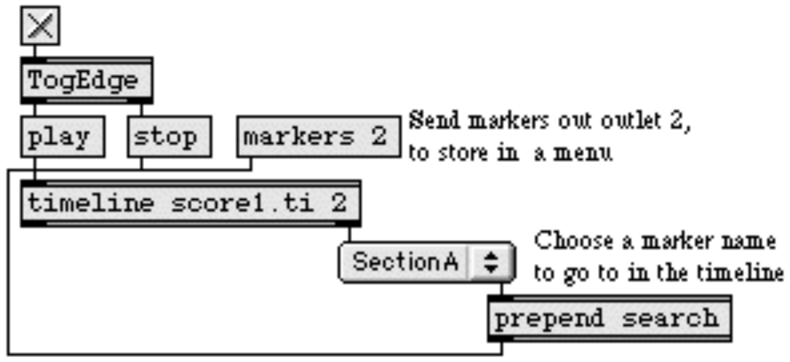
Output

any message	If the **timeline** has a positive integer argument, it will have that number of outlets. If any of its action patches (or the patch that contains the **timeline** object itself) contains a **tiout** object, then any message received in the inlet of the **tiout** is sent out the specified outlet of the **timeline** object. If the **timeline** object has no outlets, an error message will be printed in the Max window when the **tiout** object is loaded, because no message can be sent out of the **timeline** object.
(to actions)	When **timeline** receives a play message, it progresses along the timeline of events placed in its graphic editing window. When it encounters an event on the time-line, it sends that event to a specific **ticmd** object (in another patch, which has been loaded into the timeline as an action), which in turn passes the message out its own outlet.

Examples



Control a timeline's speed with **setclock**


TogEdge
play
stop
markers 2
Send markers out outlet 2, to store in a menu
timeline score1.ti 2
SectionA
Choose a marker name to go to in the timeline
prepend search

Use markers to go to specific spots on the timeline

See Also

mtr	Multi-track sequencer
setclock	Control the clock speed of timing objects remotely
thistimeline	Send messages to a **timeline**
thistrack	Send messages to a **timeline** track
ticmd	Receive messages from a **timeline**
tiout	Send messages out of a **timeline** object
Tutorial 41	Timeline of Max messages
Timeline	Creating a graphic score of Max messages

timer

Report elapsed time between two events

Input

bang	In left inlet: Starts—or *re*starts—the **timer**.
	In right inlet: Sends out the time elapsed since the **timer** was started.
clock	In left inlet: The word clock, followed by the name of an existing **setclock** object, causes the **timer** object's clock to be controlled by that **setclock** rather than by Max's internal millisecond clock. The word clock by itself sets **timer** back to using Max's regular millisecond clock.

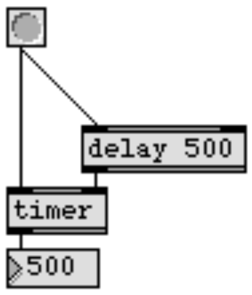
Arguments

None.

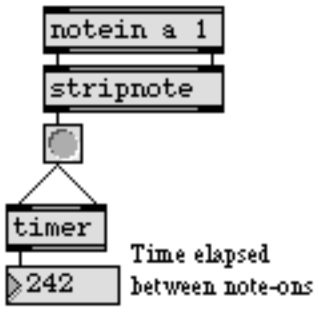
Output

float	When a bang is received in the *right* inlet, the time elapsed—in milliseconds—since the **timer** was started, is sent out the outlet.

Examples



Report time between bang *messages*



A single event can report time, then restart **timer**

See Also

clocker	Report elapsed time, at regular intervals
delay	Delay a bang before passing it on
setclock	Control the clock speed of timing objects remotely
Tutorial 20	Using the computer keyboard

tiout

Send messages out of a **timeline** *object*

Input

any message	The **tiout** object is designed to be used in an action patch. Any message received by **tiout** in an action patch is sent out an outlet of the **timeline** object that is using that action.

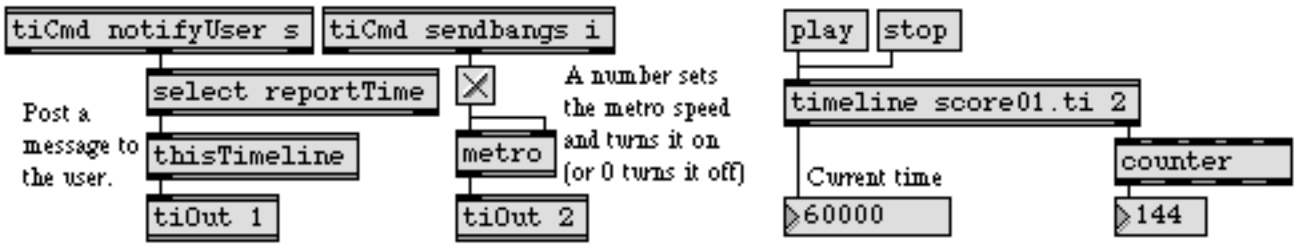
Arguments

Optional. Specifies the outlet of the **timeline** object, out of which to send messages. If no argument is present, the **tiout** object's messages are sent out outlet 1 of the **timeline** (the left outlet).

Output

(to **timeline**)	Any message received in the inlet is sent out the specified outlet of the **timeline** object that contains the **tiout** in one of its actions. If the **timeline** object has no outlets, an error message will be printed in the Max window when the **tiout** object is loaded, and no message will be sent from **tiout** to the **timeline** object.

Examples



Messages going into **tiout** *come out the specified outlet of the* **timeline** *that contains it*

See Also

ticmd	Receive messages from a **timeline**
timeline	Time-based score of Max messages
Timeline	Creating a graphic score of Max messages
Tutorial 41	Timeline of Max messages

togedge

Report zero/ non-zero transitions

Input

int	The number is stored in **togedge**. If it is not 0, and the previously stored number was 0, **togedge** sends a bang out the left outlet. If the number is 0, and the previously stored number was not 0, **togedge** sends a bang out the right outlet. Otherwise, **togedge** sends no output.
float	Ignored by **togedge**.
bang	Switches the value stored in **togedge** from 0 to non-zero, or vice versa, and reports the change by sending a bang out one of the outlets.

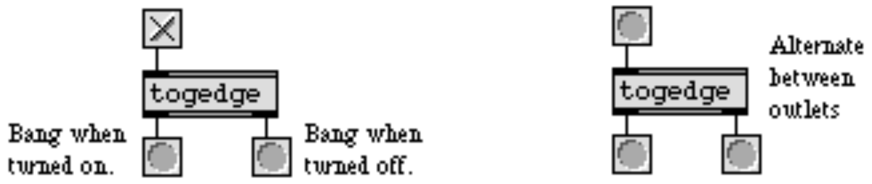
Arguments

None.

Output

bang	Out left outlet: If the stored value is changed from 0 to not 0. Out right outlet: If the stored value is changed from not 0 to 0.

Examples



Used as a detector of on/off status, or to switch back and forth between two triggers

See Also

change	Filter out repetitions of a number
led	Display on/off status in color
toggle	Switch between on and off (1 and 0)

toggle

Switch between on and off (1 and 0)

Input

int	The number is sent out the outlet. If the number is not 0, **toggle** displays an X, showing it is *on*. If it is 0, **toggle** is blank, showing it is *off*.
float	Converted to int.
bang	Switches **toggle** on if it is off; switches it off if it is on.
	A mouse click on **toggle** has the same effect as a bang in its inlet.
set	Switches the **toggle** on or off without sending anything out the outlet. The word set, followed by any non-zero number, sets toggle to on; set 0 sets it to off.
(mouse)	Clicking on a **toggle** is the same as sending it a bang message.

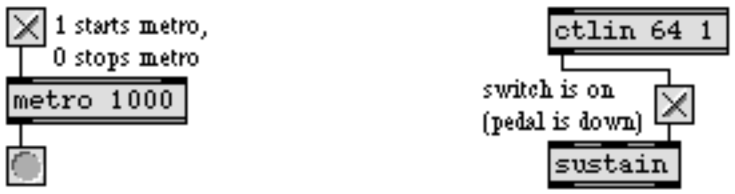
Arguments

None.

Output

int	A number received in the inlet is sent out the outlet. A bang or a mouse click sends 1 or 0 out the outlet, depending on whether **toggle** is being turned on or off.

Examples



Used as an onscreen controller, or to display the on/off status of numbers passing through

See Also

led	Display on/off status in color
matrixcrtrl	Matrix-style switch control
pictctrl	Picture-based control
radiogroup	Radio button/check box user interface object
togedge	Report zero/non-zero transitions
Tutorial 5	**toggle** and **comment**

tosymbol

Convert messages, numbers, or lists to a single symbol

Input

any message	The **tosymbol** object accepts any message, number, or list for an input, and sends a single symbol out its output. The symbol can have a maximum length of 2048 characters.
separator	The word separator specifies the separator character to be used when concatenating. The message separator with no arguments removes all spaces when creating a symbol (e.g., 1 2 3 4 becomes 1234). When used with slash or colon separators, the separator message can be used to construct pathnames (e.g., ./patches myjunk myfile becomes ./patches/myjunk/myfile). The default separator is a space.

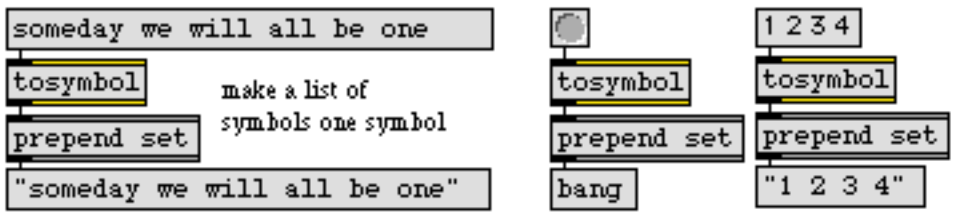
Arguments

None.

Output

symbol	A single symbol consisting of the concatenated messages, numbers, or lists. If the output symbol contains any spaces or special characters, it will be surrounded by double quotes.

Examples


someday we will all be one
tosymbol
prepend set
"someday we will all be one"
make a list of symbols one symbol
tosymbol
prepend set
bang
1 2 3 4
tosymbol
prepend set
"1 2 3 4"

Convert any input into a symbol

See Also

conformpath	Convert paths of one pathtype and/or pathstyle to another
fromsymbol	Transform a symbol into individual numbers or messages
zl	Multi-purpose list processor

touchin

Receive MIDI aftertouch values

Input

(MIDI)	**touchin** receives its input from MIDI aftertouch (channel pressure) messages received from a MIDI input device.
enable	The message enable 0 disables the object, causing it to ignore subsequent incoming MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming pitch bend messages. The word port is optional and may be omitted.
int	The number is treated as if it were an incoming MIDI aftertouch value. If there is a right outlet, 0 is sent out in lieu of a MIDI channel number. The received number is sent out the left outlet, and is not limited in the range 0 to 127.
(mouse)	Double-clicking on a **touchin** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

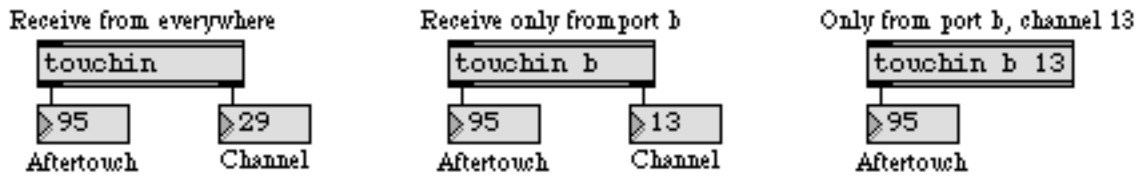
a-z	Optional. Specifies the port from which to receive incoming aftertouch messages. If there is no argument, **touchin** receives on all channels from all ports.
(MIDI name)	Optional. The name of a MIDI input device may be used as the first argument to specify the port.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to receive aftertouch messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the MIDI Setup dialog.

Output

int	If a specific channel number is included in the argument, there is only one outlet. The output is the incoming aftertouch value, from 0-127, on the specified channel and port.
	If there is no channel number specified by the argument, **touchin** will have a second outlet, on the right, which will output the channel number of the incoming aftertouch message.

touchin

Examples



Aftertouch messages can be received from everywhere, a specific port, or a specific port and channel

See Also

touchout	Transmit MIDI aftertouch messages
midiin	Output received raw MIDI data
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified
Tutorial 16	More MIDI ins and outs

touchout

Transmit MIDI aftertouch messages

Input

int	In left inlet: The number is transmitted as an aftertouch value on the specified channel and port. Numbers are limited between 0 and 127. In right inlet: The number is stored as the channel number on which to transmit the aftertouch messages.
float	Converted to int.
list	In left inlet: The first number is the aftertouch value, and the second number is the channel, of a MIDI aftertouch message, transmitted on the specified channel and port.
enable	The message enable 0 disables the object, causing it not to transmit MIDI data. The word enable followed by any non-zero number enables the object once again, even if the entire patcher window has had its MIDI disabled by an enable message to a **pcontrol** object.
port	The word port, followed by a letter a-z or the name of a MIDI input port or device, sets the port from which the object receives incoming pitch bend messages. The word port is optional and may be omitted.
(mouse)	Double-clicking on a **touchout** object shows a pop-up menu for choosing a MIDI port or device.

Arguments

a-z	Optional. Specifies the port for transmitting MIDI aftertouch messages. Channel numbers greater than 16 received in the right inlet will be *wrapped around* to stay within the 1-16 range. If there is no argument, **touchout** initially transmits out port a, on MIDI channel 1.
a-z and int	A letter and number combination (separated by a space) indicates a port and a specific MIDI channel on which to transmit aftertouch messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
(MIDI name)	Optional. The name of a MIDI output device may be used as the first argument to specify the port.
int	A number alone can be used in place of a letter and number combination. The exact meaning of the channel number argument depends on the channel offset specified for each port in the MIDI Setup dialog.

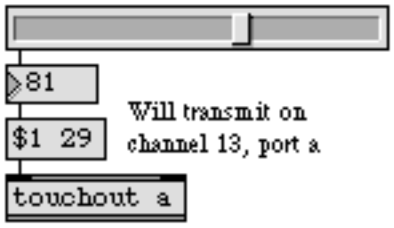
Output

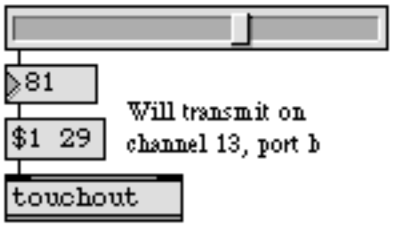
(MIDI)	There are no outlets. The output is a MIDI aftertouch message transmitted directly to the object's MIDI output port.

touchout

Transmit MIDI aftertouch messages

Examples





Letter argument transmits to only one port *Otherwise, number specifies both port and channel*

See Also

touchin	Output received MIDI aftertouch values
midiout	Transmit raw MIDI data
Using MIDI	Using Max with MIDI
Ports	How MIDI ports are specified
Tutorial 16	More MIDI ins and outs

trigger / t

Send input to many places, in order

Input

int or float	The number is sent out each outlet in the form designated by the typed-in arguments: either an int, a float, a list, a symbol (although empty), or a bang.
bang	Causes either a bang, an integer 0, a float 0., a list 0, or an empty symbol to be sent out of each outlet.
list	The list is sent out any outlet with the letter l assigned to it. Out other outlets, the list is converted and sent out as integer 0, float 0., the empty symbol "", or bang.
symbol	The word will be sent out any outlet with the letter s assigned to it. Out other outlets, the symbol is converted and sent out as integer 0, float 0., list 0, or bang.

Arguments

i, f, b, l, or s	Optional. The number of arguments determines the number of outlets. Each outlet sends out either int, float, bang, list, or symbol, depending on the arguments. If there are no arguments, there are two outlets, both of which send an int.
any message	Optional. When an int, float, or symbol is specified, the value is output as a constant.

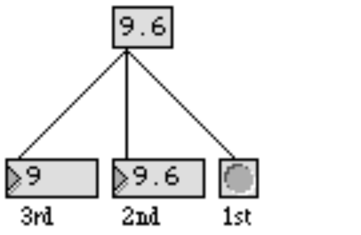
Output

int or float	A number received in the inlet is sent out each outlet, in order from right to left. The number will be converted to int, float, list, symbol, or bang before being sent out, depending on the argument that corresponds to each outlet. A symbol, list, or bang received in the inlet will be converted to integer 0 by an i outlet, and to float 0. by an f argument.
bang	Anything received in the inlet will be converted to bang before being sent out a b outlet.
list	A list received in the inlet will be sent out unchanged by an l outlet. Anything else will be converted to the single-item list 0 before being sent out.
symbol	A symbol received in the inlet will be sent out unchanged by an s outlet. Anything else will be converted to the null symbol "" before being sent out. Note: The only object that recognizes this null symbol is **print**, which valiantly prints the empty message in the Max window. Other objects will either ignore this null symbol or print an error message in the Max window.

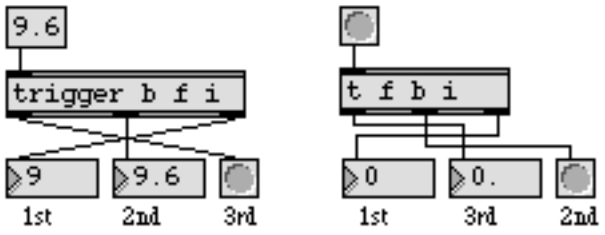
trigger / t

Send input to many places, in order

Examples



Order is normally right-to-left



Any other order can be specified by **trigger**

See Also

bangbang	Send a bang to many places, in order
message	Send any message
Tutorial 7	Right-to-left order

trough

If a number is less than previous numbers, output it

Input

int	In left inlet: If the input is less than the value currently stored in **trough**, it is stored as the new minimum value and is sent out.
	In right inlet: The number is stored in **trough** as the new minimum value, and is sent out.
float	In left inlet: Is not understood by **trough**.
	In right inlet: Converted to int.
list	In left inlet: The second number is stored as the new minimum value and is sent out, then the first number is received in the left inlet.
bang	In left inlet: Sends the currently stored minimum value out the left outlet.

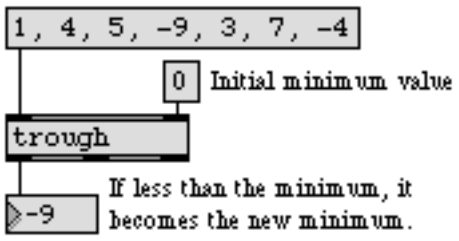
Arguments

None. The initial value stored in **trough** is 128.

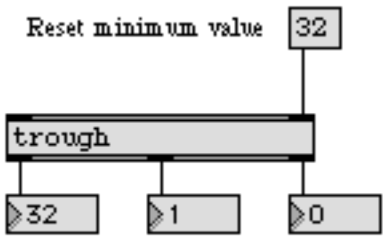
Output

int	Out left outlet: New minimum values are sent out. (Numbers received in the right inlet are *always* the new minimum value.)
	Out middle outlet: If the number received is a new minimum value, the output is 1. If the number received in the left inlet is *not* a new minimum value, the output is 0.
	Out right outlet: If the number received is a new minimum value, the output is 0. If the number received in the left inlet is *not* a new minimum value, the output is 1.

Examples



Find the smallest in a series of numbers



Number in right inlet always sets a new trough

trough

If a number is less than previous numbers, output it

See Also

minimum	Output the smallest in a list of numbers
peak	If a number is greater than previous numbers, output it
<	*Is less than*, comparison of two numbers

ubutton

Transparent button, sends a bang

Input

bang	The **ubutton** object can operate in one of two modes. When the **ubutton** is in *button* mode (the default mode), it responds to a bang in its inlet by becoming highlighted briefly and sending a bang out its left outlet. When ubutton is in *toggle* mode, a bang in its inlet causes it to become (and stay) highlighted and send a bang out its right outlet; or, if it is already highlighted, it becomes unhighlighted and sends a bang out its left outlet.
any symbol	Converted to bang.
(mouse)	In *button* mode, a mouse click on **ubutton** highlights it for as long as the mouse is held down, sending a bang out the right outlet when the mouse button is pressed down, and another bang out the left outlet when the mouse button is released. In *toggle* mode, a mouse click behaves the same as a bang. When the mouse is clicked, **ubutton** will send a 1 out the right outlet if the cursor is inside of the **ubutton** object's rectangle, and 0 if it is not. It will also send these messages when the mouse button is released. When the object is in "Track Mouse While Dragging" mode, these messages are sent continuously while the mouse button is held down after a click.
stay	The word stay, followed by a nonzero number, puts **ubutton** into *button* mode and sets it to wait for that particular number. When that number is received in the inlet, no output is sent, but **ubutton** stays highlighted until some *other* message (or a mouse click) is received. A message of stay 0 puts the **ubutton** into normal *button* mode; it no longer looks for any particular number.
int	If **ubutton** is waiting for a particular number (its *Stay-on Value*) and the incoming number matches it, the button is highlighted but nothing is sent out. If the incoming number does not match the number that **ubutton** is waiting for, the button is unhighlighted (or remains that way). If **ubutton** has a *Stay-on Value* of 0, int is the same as bang.
float	Converted to int.
dragtrack	The word dragtrack, followed by a nonzero number, enables "Track Mouse While Dragging" mode. In this mode, positional and inside/outside messages (described above for mouse clicks) are sent continuously while the mouse button is held down after a click. dragtrack 0 disables this behavior, which is off by default. Dragging the mouse will continue to generate these message pairs until the mouse button is released. Drag tracking is off by default. It can also be enabled in the **ubutton** object's Inspector.
set	If **ubutton** is in *toggle* mode, set 1 sets the **ubutton** object's toggle (highlights it) and set 0 clears the **ubutton** object's toggle (unhighlights it). Other integer arguments for set will send the number to **ubutton**, for comparison to its *Stay-on Value*, without causing any output.

toggle The word toggle, followed by a non-zero number, puts the **ubutton** in *toggle* mode. The message toggle 0 puts the **ubutton** in *button* mode.

Inspector

The behavior of a **ubutton** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **ubutton** object displays the **ubutton** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **ubutton** Inspector lets you specify the *Button Mode* (the default) or *Toggle Mode*. The *Highlight When Clicked* check box sets the mouse behavior of the ubutton object. The *Track Mouse While Dragging*" checkbox enables cursor position reporting (see the dragtrack message). Typing a nonzero number into the *Stay-on Value* box specifies the number the **ubutton** will wait for in *Button Mode*. To choose *Toggle Mode*, you must set the *Stay-on Value* to 0.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

None.

Output

bang Out 1st outlet: In *button* mode (with a *Stay-on Value* of 0), any input causes **ubutton** to flash and send a bang out the left outlet. A bang is also sent out the left outlet when the mouse button is released.

If the **ubutton** object is in *toggle* mode and is already highlighted, any input causes **ubutton** to become unhighlighted and send a bang out its left outlet.

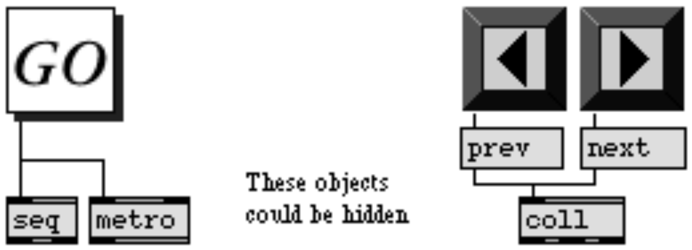
bang Out 2nd outlet: In *button* mode (with a *Stay-on Value* of 0), a mouse click sends a bang when the mouse button is pressed.

If the object is in *toggle* mode, any input causes **ubutton** to become highlighted and send a bang out the outlet. If it is already highlighted, it becomes unhighlighted and no bang is sent.

list Out 3rd outlet: When the mouse button is clicked and released, the **ubutton** object sends out a list composed of two numbers which specify the coordinates for the cursor position expressed as an offset, in pixels, from the upper left-hand corner of the **ubutton** object rectangle. If the "Track Mouse While Dragging" option is enabled using the Inspector or the dragtrack message, new coordinates will be reported as the mouse is moved until the mouse button is released.

int Out right outlet: When the mouse button is clicked and released, a 1 is sent out this outlet if the cursor is inside of the **ubutton** object's rectangular area. If the "Track Mouse While Dragging" option is enabled using the Inspector or the dragtrack message, a 0 will be output if the cursor moves outside of the **ubutton** object's rectangular area while the mouse button is pressed.

Examples



When **ubutton** *is placed on comments or pictures, they can "respond" to a mouse click*

See Also

bangbang	Send a bang to many places, in order
button	Flash on any message, send a bang
fpic	Display a picture from a graphics file
led	Display on/off status in color
matrixcrtrl	Matrix-style switch control
pictctrl	Picture-based control
radiogroup	Radio button/check box user interface object
Tutorial 19	Screen aesthetics

umenu

Pop-up menu, to display and send commands

Input

int	The number specifies a menu item to be sent out, and causes umenu to display that item. The items are numbered starting at 0.
	A menu item can also be chosen from a **umenu** with the mouse, as with any pop-up menu.
append	The word append, followed by any message, appends that message as the new last item in the menu.
autosize	The word autosize, followed by a 1 or 0, turns sizing the pop-up menu to the width of the longest item on or off. If autosize is off, the width of the menu is the width of the object's rectangle.
bang	Sends out the currently displayed menu item.
brgb	The word brgb, followed by three numbers between 0 and 255, sets the color of the **umenu** object in RGB format. The default is 221 221 221.
checkitem	The word checkitem, followed by an item number and 1 or 0, places (1) or removes (0) a check mark next to the item number.
clearchecks	The word clearchecks removes check marks for all items.
clear	Removes all items from the **umenu**.
color	The word color, followed by a number between 0 and 15, sets the foreground (text) color to the standard preset color specified by the number.
delete	Followed by a number of an item, deletes that item from the **umenu**.
evalitemtext	The word evalitemtext, followed by a 1 or 0, turns Evaluate Item Text mode on or off. When on, the message represented by the current item's text is sent out the right outlet when the menu's value is changed either by message or the user clicking on it.
frgb	The word rgb, followed by three numbers between 0 and 255, sets the text color of the **umenu** object in RGB format. The default is 0 0 0.
labelclick	The word labelclick, followed by a 1 or 0, turns Label Click mode on or off. In this mode, when the object is in Label mode, you can click in the object's rectangle and the current value of the menu is sent out the left outlet. In addition, the text of the current item is shown underlined.
maxitems	The word maxitems, followed by the number, sets the maximum number of menu items of the **umenu**, in the same way as the *Maximum number of items* setting in

	the **umenu** object's Inspector (see *Inspector*, below). The default is 64, and the maximum is 2000.
mode	The word mode, followed by the number 1, 2, or 3, sets the appearance and behavior of the **umenu**, in the same way as the *Mode* setting in the **umenu** object's Inspector (see *Inspector*, below). mode 1 is the normal pop-up menu style, mode 2 is a scrolling menu style, and mode 3 is a label instead of a menu.
rgb2	The word rgb2, followed by three numbers between 0 and 255, sets the upper frame light color (i.e., the "lit" part of a 3D menu item) of the **umenu** object's menu item in RGB format. The default is 255 255 255.
rgb3	The word rgb3, followed by three numbers between 0 and 255, sets the upper frame dark color (i.e., the "shaded" part of a 3D menu item) of the **umenu** object's menu item in RGB format. The default is 221 221 221.
rgb4	The word rgb4, followed by three numbers between 0 and 255, sets the lower frame light color (i.e., the "lit" part of a 3D menu item) of the rectangle that outlines the **umenu** object's menu item in RGB format. The default is 170 170 170.
rgb5	The word rgb5, followed by three numbers between 0 and 255, sets the lower frame dark color (i.e., the "shaded" part of a 3D menu item) of the **umenu** object's menu item in RGB format. The default is 119 119 119.
rgb6	The word rgb6, followed by three numbers between 0 and 255, sets the color of the "corner dots" of the **umenu** display area in RGB format. If you are using a **umenu** object on a colored background or in front of a panel, you should set this color to match the background object color. The default is 187 187 187.
set	The word set, followed by a number or symbol, specifies a menu item to be displayed by **umenu**, but does not send it out the outlet. If the set argument is a symbol, set searches for a menu item which begins with the symbol.
setcheck	(Macintosh only) The word setcheck, followed by a number between 0 and 255, sets the character used to be the check mark. setcheck 0 uses the default character.
setitem	The word setitem, followed by an item number and any message, sets the specified menu item to that message.
setrgb	The word setrgb, followed by six numbers between 0 and 255 that specify RGB values, uses the first three numbers to set the foreground (text) color and the second three numbers to set the background (fill) color.
showchecked	This message operates as follows. If the currently displayed item is checked, do nothing. Otherwise, starting at the first item in the menu, find one that is checked and set the menu to display that item. If there isn't one, do nothing.

symbol Identical to the set message with a symbol argument, except that the found item number is sent out (and the text of the item is sent out the right outlet, if the *Evaluate Item Text* feature is enabled).

Inspector

The behavior of a **umenu** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **umenu** object displays the **umenu** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

Enter the items which you want to appear on the menu in the *Menu Text* box, separated by commas. The the *Maximum Items* box lets you specify the maximum number of menu items. You nay have any number of menu items from 32 to 2000 (the default is 64). The pop-up *Mode* menu lets you specify the appearance and behavior of the **umenu** object's user interface. *Normal* (the default) is the standard pop-up menu, allowing you to see all the menu items at once by clicking and holding the mouse button. *Scrolling* mode lets you scroll through the individual menu items by dragging the mouse up or down, displaying one item at a time; "Label" shows the text of the selected menu item with no border around it, and does not respond to the mouse. If *Auto Size* is checked, the width of the **umenu** object's pop-up menu will be adjusted to fit the width of the longest item. If *Evaluate Item Text* is checked, the text of the menu item will be sent as a message out the right outlet when the item is selected.

The *Color* option lets you use a swatch color picker or RGB values used to display the **umenu** text and its background. *Text* sets the color for the message displayed (default 0 0 0), and *Background* sets the color for the message area in which the hint appears (default 221 221 221). The *Upper Frame Light*, *Upper Frame Dark*, *Lower Frame light*, and *Lower Frame Dark* attributes are used to set the "lit" and "shaded" edges of the menu item. The default settings are 255 255 255 for upper frame light, 221 221 221 for upper frame dark, 170 170 170 for lower frame light, and 119 119 119 for lower frame dark. *Corner Dots* is used to set the color of the corner area of the **umenu** item's display area. The default is 187 187 187.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

The font and size of a **umenu** can be changed with the Font menu.

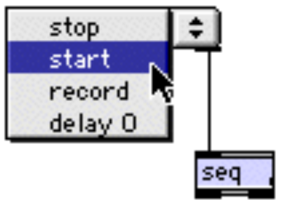
Arguments

None.

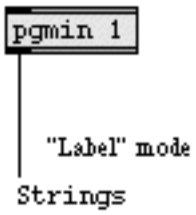
Output

int	Out left outlet: The number of the selected menu item is sent out. Menu items are numbered beginning with 0.
anything	Out right outlet: If *Evaluate Item Text* has been checked in the Inspector, the text of the selected menu item is sent out as a message.

Examples



Used to send commands



…or to display text associated with numbers received

See Also

coll	Store and edit a collection of different messages
Tutorial 37	Data structures

universal

Send a message to all instances of the same class in a patcher

Input

class symbol	The universal objects expects as input a symbol that names an object class (for example, **table** or **dspstate~**), followed by a message selector and any number of arguments for that message. The message and its arguments (if any) are sent to all instances of the class within the same patcher (and possibly its subpatchers).
sendmessage	To send messages to certain objects whose class names are also reserved Max message names (such as int and float), you need to start the message with the send-message message. **sendmessage** can be used with any class.

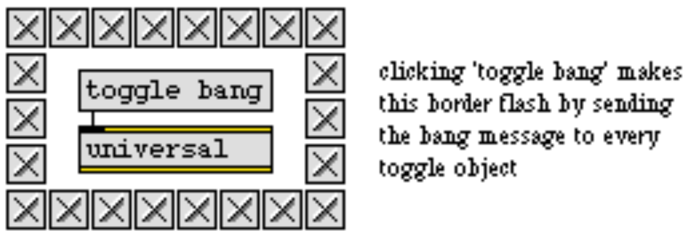
Arguments

int	Optional. If a 1 is present as an argument, **universal** will send messages it receives to objects of the specified class in subpatchers of its patcher as well as in the patcher containing the **universal** object.

Output

None. The object has no outlets, but objects receiving the message(s) it sends may have some form of output from their outlets. However, the order in which the message is sent to various objects is not guaranteed. This is also true when using the **send** and **receive** objects.

Examples


toggle bang
universal
clicking 'toggle bang' makes this border flash by sending the bang message to every toggle object

Send a message to all objects of the same class at once

See Also

forward	Send remote messages to a variety of objects
receive	Receive messages without patch cords
send	Send messages without patch cords
value	Share a stored message with other objects

unpack

Break a list up into individual messages

Input

list	Each item in the list (up to the number of outlets) is sent out the outlet corresponding to its position in the list.
int	The number is sent out the left outlet.
float	Converted to int, unless the left outlet was initialized with a float argument. The number is sent out the left outlet.
symbol	The symbol is sent out the left outlet. If the left outlet was not initialized with a symbol argument, 0 is sent out the outlet. Symbol arguments allow symbols to pass through, and change numbers to the empty symbol (““).
bang	Causes each stored item of a list received in the inlet to sent out the corresponding outlet.

Arguments

anything	Optional. The number of outlets is determined by the number of arguments. The arguments can be any combination of ints, floats, and symbols. The argument specifies the output of the **unpack** object's outlet; the input type is forced to the outlet type (e.g., outlets that correspond to int or float arguments will always output that type of number, converting the input items as necessary). If no argument is typed in, **unpack** will have two int outlets. Symbol arguments allow symbols to pass through, and change numbers to the empty symbol (““).

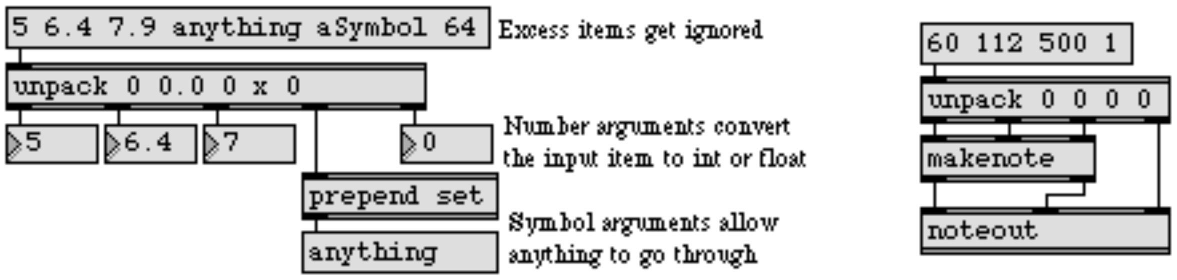
Output

int	Each item of the list received in the inlet is sent out the corresponding outlet. The first item in the list is sent out the leftmost outlet, and so on. If an outlet has been initialized with an int argument, then a float or a symbol will be converted to int before being sent out that outlet. (A symbol is converted to 0.)
float	If the outlet has been initialized with a float argument, then an int or a symbol from the input list will be converted to float before being sent out that outlet. (A symbol is converted to 0.0.)
symbol	A symbol in the input list will be sent out the corresponding outlet if that outlet has been initialized with a symbol argument. If the outlet has been initialized with an int or a float, the symbol will be converted to 0 or 0.0.

unpack

Break a list up into individual messages

Examples


5 6.4 7.9 anything aSymbol 64
Excess items get ignored
unpack 0 0.0 0 x 0
5
6.4
7
0
Number arguments convert the input item to int or float
prepend set
anything
Symbol arguments allow anything to go through
60 112 500 1
unpack 0 0 0 0
makenote
noteout

Each item in a list can be sent to a different place

See Also

iter	Break a list up into a series of numbers
pack	Combine numbers into a list
spray	Distribute an integer to a numbered outlet
zl	Multi-purpose list processor
Tutorial 30	Number groups

urn

Generate random numbers without duplicates

Input

bang	In left inlet: Sends out a previously unchosen random number from 0 to one less than the specified maximum limit.
clear	In left inlet: Clears the list of already chosen numbers.
int	In right inlet: Clears the list of already chosen numbers, and specifies the number of possible values for the random number generator. The random numbers will range from 0 to one less than this maximum limit.
seed	In left inlet: The word seed, followed by a number, provides a "seed" value for the random generator, which causes a specific (reproducible) sequence of pseudo-random numbers to occur. The number 0 uses the time elapsed since system start-up (an unpredictable value) as the seed, ensuring an unpredictable sequence of numbers. This unpredictable seed is used by default when the **urn** object is created. However, once all numbers have been chosen, the sequence will repeat. Therefore, in order to achieve a non-repeating sequence of numbers, you will need to send the **urn** object the seed 0 message each time you send it the clear message.

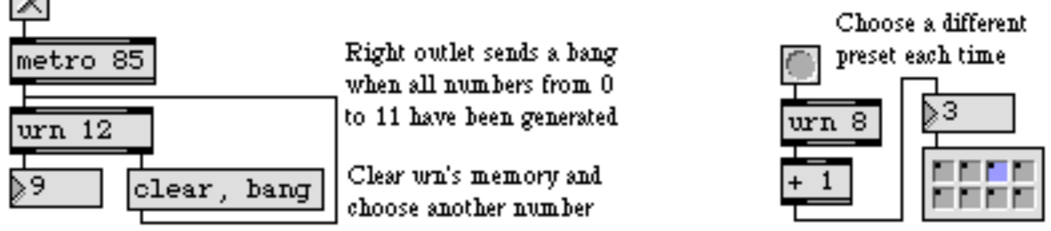
Arguments

int	Optional. The number of possible values for the random number generator. If no argument is typed in, there will be only 1 possible number.

Output

int	Out left outlet: If there are numbers within the current range that have not been sent out since the last clear message was received, **urn** generates a random number between 0 and one less than the maximum.
bang	Out right outlet: When all numbers in the current range have been generated, **urn** sends a bang out the right outlet instead of a number out the left outlet.

Examples


metro 85
urn 12
9
clear , bang
Right outlet sends a bang when all numbers from 0 to 11 have been generated
Clear urn's memory and choose another number
Choose a different preset each time
urn 8
+ 1
3

Choose random numbers without repeating a choice

See Also

decide	Choose randomly between on and off (1 and 0)
drunk	Output random numbers in a moving range
random	Generate a random number

uslider

Output numbers by moving a slider onscreen

Input

int	The number received in the inlet is displayed graphically by **uslider**, and is passed out the outlet. Optionally, **uslider** can multiply the number by some amount and add an offset to it, before sending it out the outlet.
(mouse)	The **uslider** will also send out numbers in response to mouse clicking or dragging.
float	Converted to int.
bang	Sends out the number currently stored in **uslider**.
color	The word color, followed by a number from 0 to 15, sets the color of the center portion of the **uslider** to one of the object colors which are also available via the Color command in the Object menu.
local	The word local, followed by a non-zero number, enables object response to mouse clicks (the default). The message local 0 disables the object's response to the mouse; the **uslider** object will respond only to input in its inlet and ignore all mouse clicks.
min	The word min, followed by a number, sets value that will be added to the **uslider** object's value before it is sent out the outlet. The default is 0.
mult	The word mult followed by a number, specifies a multiplier value. The **uslider** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default value is 1.
resolution	The word resolution, followed by a number, sets the sampling interval in milliseconds. This controls the rate at which the display is updated as well as the rate that numbers are sent out the **uslider** object's outlet.
set	The word set, followed by a number, resets the value displayed by **uslider**, without triggering output.
size	The word size, followed by a number, sets the range of the **uslider** object. The default value is 128. Setting the size to 1 disables the **uslider** visually (since it can only display one value). Any specified size less than 1 will be set to 2.

Inspector

The behavior of a **uslider** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector from the Windows menu, selecting any **uslider** object displays the **uslider** Inspector in the floating window. Selecting an object and choosing Get Info… from the Object menu also displays the Inspector.

The **uslider** Inspector lets you enter a *Slider Range* value. Numbers received in the inlet are automatically limited between 0 and the number 1 less than the specified range value. The default range value is 128. You can specify an *Offset* value which will be added to the number, after multiplication. The default offset value is 0. The **uslider** Inspector also lets you specify a *Multiplier*. The **uslider** object's value will be multiplied by this number before it is sent out the outlet. The multiplication happens before the addition of the Offset value. The default multiplier value is 1.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing Undo Inspector Changes from the Edit menu while the Inspector is open.

Arguments

The range of **uslider** is set by selecting it (when the patcher window is unlocked) and choosing Get Info… from the Object menu. Numbers received in the inlet are automatically limited between 0 and the number 1 less than the specified range.

The Inspector also provides a *Multiplier*—by which all numbers will be multiplied before being sent out, and an *Offset*—which will be added to the number, after multiplication. A newly created **uslider** has a range of 128, a multiplier of 1, and an offset of 0.

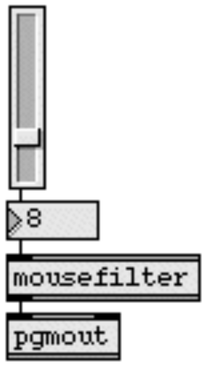
Output

int Numbers received in the inlet, or produced by clicking or dragging on **uslider** with the mouse, are first multiplied by the multiplier, then have the offset added to them, then are sent out the outlet.

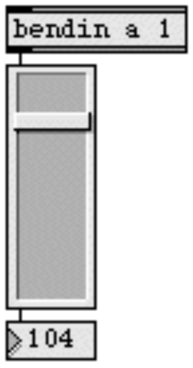
uslider

Output numbers by moving a slider onscreen

Examples



Produce output by dragging onscreen...



or use to display numbers passing through

See Also

dial	Output numbers by moving a dial onscreen
hslider	Output numbers by moving a slider onscreen
kslider	Output numbers from a keyboard onscreen
pictctrl	Picture-based control
pictslider	Picture-based slider
rslider	Display or change a range of numbers
slider	Output numbers by moving a slider onscreen
Tutorial 14	Sliders and dials
Tutorial 14	Sliders and dials

uzi

Send a specific number of bang *messages*

Input

bang	In left inlet: Begins sending out bang messages as fast as possible, one after another. The number of bang messages to send is determined by the last number received in either inlet.
int	In left inlet: Sets the number of bang messages to send, then begins sending them out as fast as possible, one after another.
	In right inlet: Sets the number of bang messages to send, without causing output.
pause	In left inlet: Causes **uzi** to stop in the midst of sending its output. (Since **uzi** sends its output as fast as possible, this message must be triggered in some way by the output of **uzi** itself.) **uzi** keeps track of how many bang messages it has sent, and if it receives the pause message before sending out all its bang messages, it can then be caused to send out the rest of its bang messages with a resume or continue message.
break	Same as pause.
resume	In left inlet: If **uzi** has been stopped by a pause message in the midst of sending its output, resume causes it to send out the rest of its output.
continue	Same as resume.

Arguments

int	Optional. Sets an initial number of bang messages to be sent out in response to a bang in the left inlet. If no argument is present, **uzi** is initially set to send out one bang.

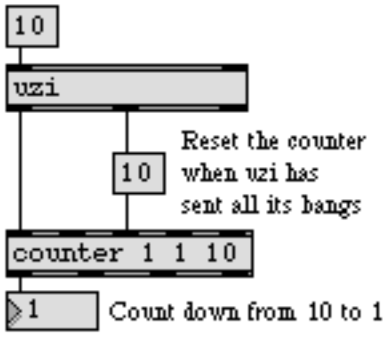
Output

bang	Out left outlet: When **uzi** receives a bang or int in its left inlet, a certain number of bang messages are sent out as fast as possible, one after another. The number of bang messages is determined by the most recent number received in either inlet.
	Out middle outlet: After the last bang is sent out its left outlet, **uzi** sends one bang out its middle outlet. This can be used as a signal that all the bang messages have been sent, much like the "carry" outlet on the **counter** object.
int	Out right outlet: The number of each bang is sent out. Numbering begins from 1 each time an int or bang is received in the left inlet. If **uzi** is being restarted with a resume or continue message, numbering begins wherever it left off.

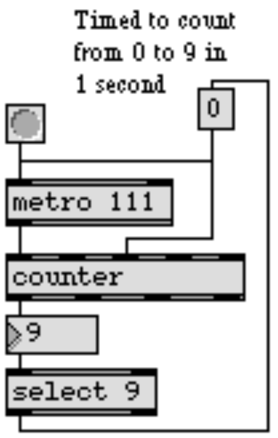
uzi

Send a specific number of bang *messages*

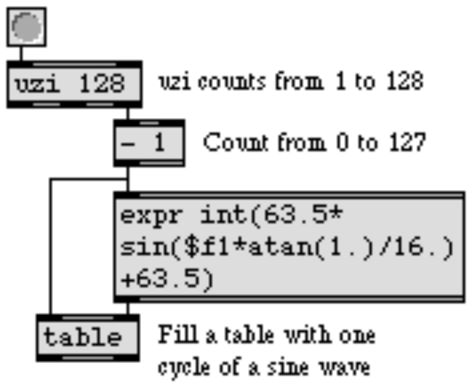
Examples



Count as fast as possible using **uzi**



Count at a specific rate, not using **uzi**


uzi 128
uzi counts from 1 to 128
- 1
Count from 0 to 127
expr int(63.5* sin($f1*atan(1.)/16.) +63.5)
table
Fill a table with one cycle of a sine wave

Use **uzi** *to perform many calculations quickly*

See Also

counter	Count the bang messages received, output the count
line	Output numbers in a ramp from one value to another
metro	Output a bang message at regular intervals

value / v

Share a stored number with other objects

Input

any message	The message is stored, to be shared by all other **value** objects with the same name, even if they are in another patch. A message received in any other **value** object that has the same name will change the stored value.
bang	Sends out the stored message.
(mouse)	Double-clicking on a **value** object opens all windows containing **value** objects with the same name.
send	The word send, followed by the name of a **receive** object, sets the value object to send its stored message to all **receive** objects with that name in response to a bang message.

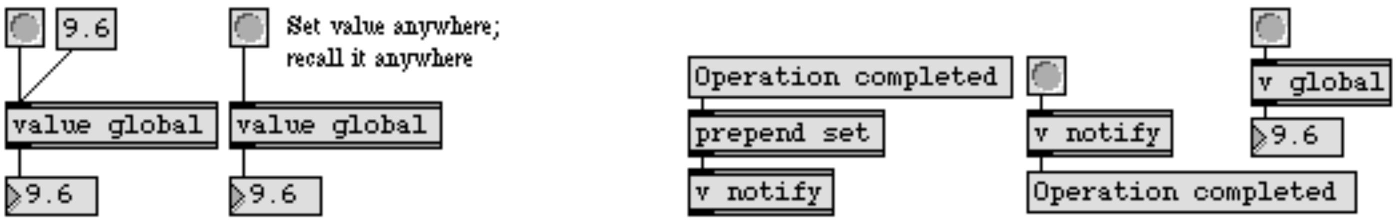
Arguments

symbol	Obligatory. Gives a name to **value**.
any message	Optional. Additional arguments after the naming symbol initialize the contents of **value**. If no additional arguments are present, **value** contains nothing.

Output

any message	A bang in the inlet causes the stored message to be sent out.

Examples



One value (or any type of message) is shared between all **value** *objects that share the same name*

See Also

float	Store a decimal number
int	Store an integer value
pv	Share variables specific to a patch and its subpatches
pvar	Connect to a named object in a patcher
send	Send messages without patch cords
receive	Receive messages without patch cords
Tutorial 24	**send** and **receive**

vdp

Control a videodisk player through a serial port

The **vdp** object works with serially-controlled videodisk players (remember them?) that are compatible with the Pioneer 4200 or 8000 standard. Each command received by the **vdp** object sends a stream of numbers out the object's left outlet, intended to be connected to the **serial** object. The description of each command below discusses what effect the command has on the player, not the exact character stream sent by **vdp**.

Because videodisc players have relatively buffer-less serial interfaces, **vdp** places each command it receives in a queue, and sends it out only when the player has finished executing its most recent command. This "feature" may cause a delay between the time a command is sent to the **vdp** object and the time it is actually sent out the serial port.

Any message received in the right inlet will behave exactly as if it had been received in the left inlet, except that it will be put at the front of the queue, to be the very next command sent out to the player.

Input

clear — In left inlet: Removes any pending commands from the queue and resets the object.

control — In left inlet: The word control, followed by a number, tells the videodisc player to perform one of the following operations:

Number	*Operation*
0	Initialize and reset player
1	Eject disk
2	Audio off
3	Audio 1 on
4	Audio 2 on
5	Stereo on
6	Picture on
7	Picture off
8	Display frame numbers on
9	Display frame numbers off
11	Frame access mode
12	Time access mode
13	Chapter access mode

fps — In left inlet: Sets the playing speed. The fps message is followed by a number (frames per second) or an adjective. The following adjectives and numbers are equivalent (at least for the Pioneer 4200):

slowest	1
slower	10
slow	15
normal	30
fast	60
faster	90
fastest	120

frame — In left inlet: Asks the player what its current frame number is and sends the response (received in the middle inlet) out the middle-right outlet.

play — In left inlet: With no arguments, play starts playing at the current speed from the current location to the end of the disk (or until the player receives another command). With one argument (a frame number), play searches to the specified frame number and begins playing to the end of the disk. With two arguments, play searches to the location specified by the first number and plays until the disc reaches the second frame number.

int — In left inlet: Same as play from a specified frame number to the end of the disc.

In middle inlet: **vdp** expects responses from the player to be fed from the **serial** object into its middle inlet. When **vdp** sees "received" (the letter R followed by the return character) from the player, it sends the next command from its queue of pending commands. The example shows how to connect the **vdp** and **serial** objects together.

scan — In left inlet: Initiates a "fast forward" or "rewind" operation. scan forward moves forward, scan backward moves backward.

search — In left inlet: The first argument indicates a frame number to search to. The second, optional argument, if non-zero, instructs the player to keep the picture on while searching. If searching a great distance from the current location, the player may not be able to keep from blanking the screen. Once the player arrives at the desired frame, it will display the (still) image from that frame.

step — In left inlet: Followed by -1, step pauses the player (if playing) and displays the previous frame. Followed by 1, step pauses the player (if playing) and displays the next frame.

stop — In left inlet: Pauses the player.

cmd — In left inlet: The cmd message can be used to send "primitive" commands consisting of ASCII codes to the video disk player. Commands usually consist of two-letter codes preceded by numeric arguments. For example, searching to frame 5000 could be accomplished with the message cmd 5000 SE. Refer to the owner's manual of your player for details. The cmd message is particularly useful with the Pioneer 8000 player, since it has a number of special features not supported by the regular messages of the **vdp** object.

setskip In left inlet: Followed by a number, sets the number of frames to jump (forward or backward) from the current frame location when using the skip message.

skip In left inlet: Followed by -1, skips backward by a number of frames specified in the setskip message. Followed by 1, skips forward by a number of frames specified in the setskip message.

Arguments

None.

Output

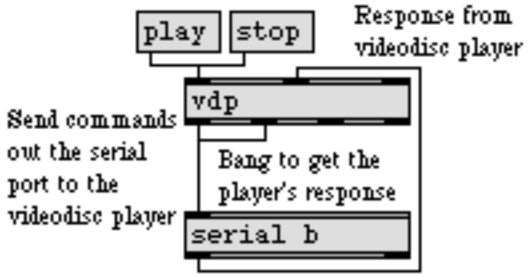
int Out left outlet: A stream of characters, coded instructions to the videodisc player, for each command. These numbers are intended to be sent to the left inlet of a **serial** object.

bang Out middle-left outlet: After sending a command out its left outlet, **vdp** begins "polling" the **serial** object for a response from the player by sending bang messages out this outlet approximately every 20 milliseconds, until **vdp** receives a "received" signal from the player in its right inlet. (A bang sent to a **serial** object causes any characters received in that serial port to be sent out the **serial** object's outlet.)

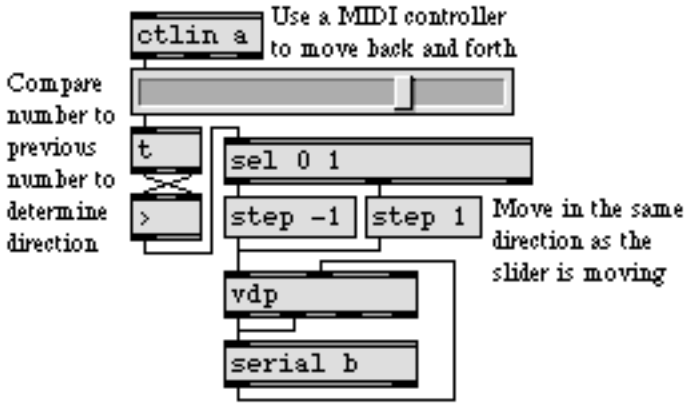
int Out middle-right outlet: Current frame number, received from the player in response to a frame message.

int Out right outlet: Not implemented.

Examples


play
stop
Response from videodisc player
vdp
Send commands out the serial port to the videodisc player
Bang to get the player's response
serial b

Basic configuration of **vdp** *and* **serial** *objects*


ctlin a
Use a MIDI controller to move back and forth
Compare number to previous number to determine direction
t
>
sel 0 1
step -1
step 1
Move in the same direction as the slider is moving
vdp
serial b

"Scrubbing" with a slider or MIDI controller

See Also

serial	Send and receive characters from serial ports and cards
	Pioneer 4200 operation manual
	HyperCard Interactive Video Toolkit documentation

vexpr

Evaluate a math expression for a list of different inputs

The **vexpr** object behaves exactly like the **expr** object, except for the way in which it handles lists. See **expr** for a full description.

Input

list	**vexpr** is designed to receive a list in each inlet. The items of each list are used individually, in order from left to right, to replace the changeable argument in a series of evaluations of the expression. When a list is received in the left inlet, the expression is first evaluated using the first item of each list, then using the second item of each list, etc. The series of results of these evaluations is then sent out as a list.
int, or float	An int or float received in any inlet is treated as a single-item list.
bang	In left inlet: Evaluates the expression and sends out the results, using the most recently received lists of numbers.
scalarmode	In left inlet: The word scalarmode, followed by a non-zero number, sets the scalar mode of operation. In scalar mode, sending a list of length 1 (i.e., a single value) will cause that value to be applied to each element of the other list. The message scalarmode 0 disables scalar mode.

Arguments

Obligatory. See **expr**.

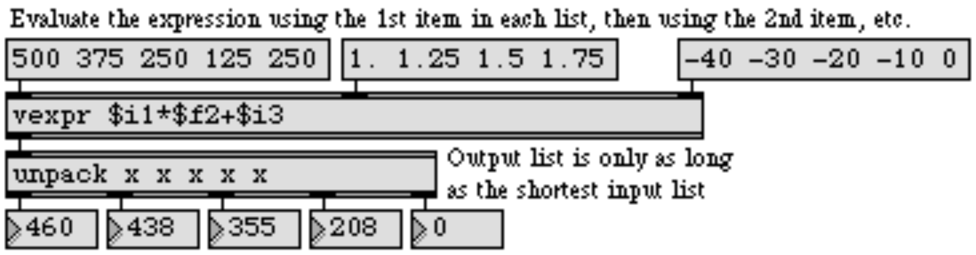
Output

list	When a list is received in the left inlet, **vexpr** uses the first item of the lists it has received in each of its different inlets, puts those items in place of the changeable arguments in the expression, and evaluates the expression. It then does the same with the second item in each list, and so on until it has used the last item of the shortest list. It then sends out all of the different results as a single list.
int	If the input in one of the inlets was a single number rather than a list, and the expression is evaluated as an integer value, then a single result is sent out as an int rather than a list.
float	If the input in one of the inlets was a single number rather than a list, and the expression is evaluated as a float value, then a single result is sent out as a float rather than a list.

vexpr

Evaluate a math expression for a list of different inputs

Examples


Evaluate the expression using the 1st item in each list, then using the 2nd item, etc.
500 375 250 125 250
1. 1.25 1.5 1.75
-40 -30 -20 -10 0
vexpr $i1*$f2+$i3
unpack x x x x x
Output list is only as long as the shortest input list
460
438
355
208
0

Perform the same calculation on a whole list of input values

See Also

expr	Evaluate a mathematical expression
Tutorial 38	**expr** and **if**

xbendin

Interpret extra precision MIDI pitch bend values

Input

int	The numbers are individual bytes of a MIDI message stream, received from an object such as **midiin** or **seq**. MIDI pitch bend messages are recognized by **xbendin**, and the pitch bend data is sent out in full precision.

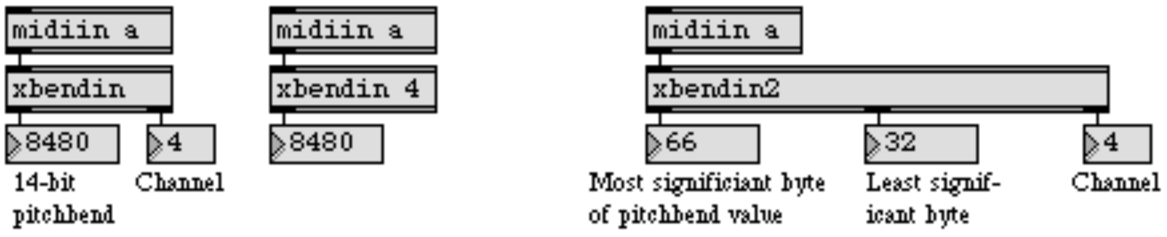
Arguments

int	Optional. The number specifies a MIDI channel on which to recognize pitch bend messages. If there is no argument, **xbendin** recognizes pitch bend messages on all channels, and the channel number is sent out the extra outlet on the right.
xbendin2	Optional. Normally, **xbendin** sends pitch bend values out the left outlet as 14-bit values. If the object is called **xbendin2**, however, there will be an additional outlet. The most significant data byte of the message is sent out the leftmost outlet, and the least significant data byte is sent out the second outlet.

Output

int	The pitch bend value is sent out the left outlet of **xbendin** as a single 14-bit value. If the object is called **xbendin2**, there is an additional outlet. The most significant 7 bits are sent out the leftmost outlet, and the least significant (extra precision) 7 bits are sent out the second outlet. If there is no channel number specified as an argument (omni on), **xbendin** will have an extra outlet on the right, which will output the channel number of the incoming pitch bend message.

Examples



Pitch bend values are sent out as a single number or as two separate bytes

See Also

bendin	Output received MIDI pitch bend messages
midiin	Output received raw MIDI data
xbendout	Format extra precision MIDI pitch bend messages
Tutorial 34	Managing raw MIDI data
MIDI	MIDI overview and specification

xbendout

Format extra precision MIDI pitch bend messages

Input

int	In left inlet: The number is a 14-bit pitch bend value to be formatted into a complete MIDI pitch bend message by **xbendout**.
	In right inlet: The number is stored as the MIDI channel for the pitch bend message sent out by **xbendout**. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.
list	The first number is a 14-bit pitch bend value, and the second number is the channel. Both numbers are stored and are formatted into a MIDI pitch bend message which is sent out the outlet.
bang	Sends out a MIDI pitch bend message using the numbers currently stored in **xbendout**.

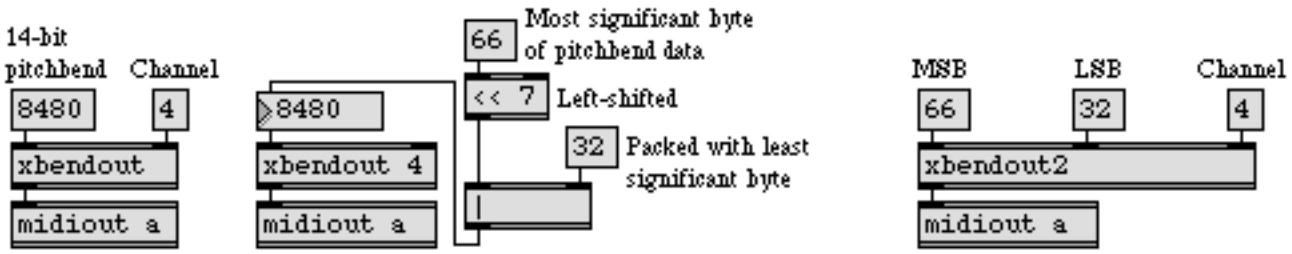
Arguments

xbendout2	If the object is called **xbendout2**, there will be three inlets. The most significant byte of the pitch bend message is received in the left inlet, and the least significant (extra precision) byte is received in the middle inlet.
int	Optional. The number sets an initial value for the MIDI channel of the pitch bend messages. If there is no argument, the initial channel number is 1.

Output

int	When a pitch bend value is received in the left inlet, the complete MIDI pitch bend message is sent out the outlet, byte-by-byte.

Examples



14-bit pitch bend value is formatted into a MIDI message, which is sent out byte-by-byte

See Also

bendout	Transmit MIDI pitch bend messages
midiout	Transmit raw MIDI data
xbendin	Interpret extra precision MIDI pitch bend messages
Tutorial 34	Managing raw MIDI data
MIDI	MIDI overview and specification

xnotein

Interpret MIDI note messages with release velocity

Input

int — The numbers are individual bytes of a MIDI stream from **midiin**. Whereas a note-on with a velocity of 0 is most commonly used to indicate a note-off, **xnotein** also recognizes the MIDI note-off command, and outputs its release velocity.

Arguments

int — Optional. Specifies a channel number on which to look for incoming MIDI note-on and note-off messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range. If there is no argument, **xnotein** recognizes note-on and note-off messages on all MIDI channels, and the channel number of the message is sent out the rightmost outlet.

Output

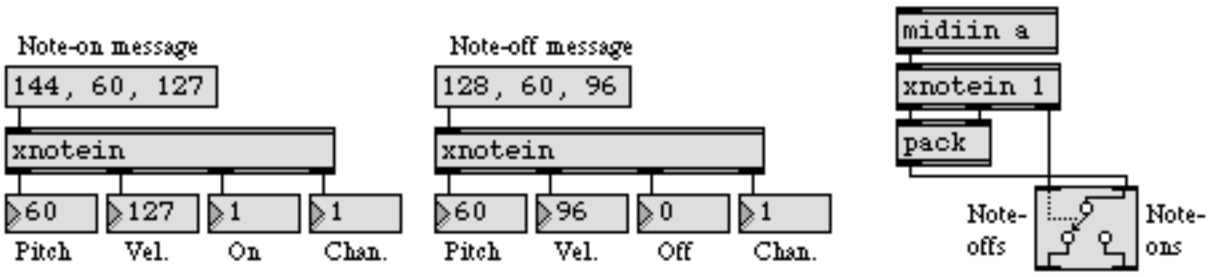
int — Out left outlet: The pitch value of the incoming note-on or note-off message.

Out 2nd outlet: The key-down or key-up velocity of a note-on or a note-off message.

Out 3rd outlet: The number is the indicator of whether the incoming MIDI message is a note-on or a note-off. If the incoming message is a note-on, the output is 1. If the incoming message is a note-off—or a note-on with a velocity of 0—the output is 0.

If no channel number is specified as an argument, **xnotein** has a 4th outlet on the right. The channel number of incoming messages is sent out the rightmost outlet.

Examples



Both note-on and note-off messages are interpreted, with a key-down or key-up velocity

See Also

notein	Output received MIDI note messages
midiin	Output received raw MIDI data
xnoteout	Format MIDI note messages with release velocity
Tutorial 34	Managing raw MIDI data
MIDI	MIDI overview and specification

xnoteout

Format MIDI note messages with release velocity

Input

int — In left inlet: The number is used as the pitch value for a note-on or note-off message, and the message is sent out the outlet byte-by-byte.

In left-middle inlet: The number is stored as the velocity for either a note-on or a note-off message. If no number has been received yet, the velocity for note-ons is 64, and the velocity for note-offs is 0.

In right-middle inlet: The number is stored as the indicator of whether outgoing messages should be note-ons or note-offs. If the number is not 0, **xnoteout** will send out a note-on message. If the number is 0, **xnoteout** will send out a note-off message with a release velocity. If no number has been received yet, it is initially 1 (note-on).

In right inlet: The number is stored as the channel for the MIDI message sent out by **xnoteout**. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range.

float — In left inlet: Is not understood by **xnoteout**.

In other inlets: Converted to int.

list — The first number is the pitch value, the second number is the velocity, the third number is the note-on/note-off indicator (non-zero for note-on, 0 for note-off), and the fourth number is the channel. The numbers are stored by **xnoteout**, and a MIDI note-on or note-off message is sent out.

bang — Sends out a MIDI message using the numbers currently stored in **xnoteout**.

Arguments

int — Optional. Sets an initial value for the MIDI channel of the outgoing messages. Channel numbers greater than 16 will be *wrapped around* to stay within the 1-16 range. If there is no argument, the initial channel number is 1.

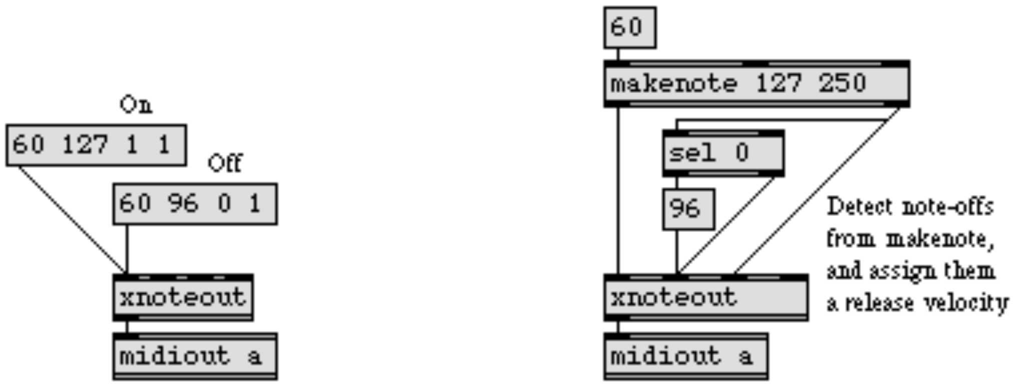
Output

int — When a pitch value is received, a complete MIDI note-on or note-off message is sent out the outlet, byte-by-byte. Whereas a note-on with a velocity of 0 is most commonly used to indicate a note-off, **xnoteout** sends out the MIDI note-off command and uses the specified velocity as a *release* velocity.

xnoteout

Format MIDI note messages with release velocity

Examples



The numbers are formatted into a MIDI note-on or note-off message, which is sent out byte-by-byte

See Also

noteout	Transmit MIDI note messages
midiout	Transmit raw MIDI data
xnotein	Interpret MIDI note messages with release velocity
Tutorial 34	Managing raw MIDI data
MIDI	MIDI overview and specification

zl

The **zl** object performs several kinds of list processing functions. You set the function with a keyword argument, and can change the function performed with the mode message. The behavior of the **zl** object's inlets and outlets and the types of messages they expect or process varies according to the mode of the **zl** object. For brevity in the discussion that follows, we refer to any Max message as a list including single elements such as int, symbol, and float and messages that begin with a symbol (a Max list is a message that begins with a number).

Input

mode The word mode, followed by one of the symbols group, iter, join, len, reg, rev, rot, sect, slice, or union, sets the current mode of the **zl** object. For some modes of operation, A list received in the left inlet may be used as an argument to specify the functionality of a given mode. The items of messages that are not long enough to send out are added to the length of the stored list. Once the stored list is long enough, it is sent out the left outlet.

mode ecils is used to divide a list into two lists. This mode takes an additional number argument which specifies the size, in elements, of a list. This value can also be specified as an input in the right inlet in this mode. A list received in the left inlet will be split into two lists—the first list contains the number of items specified by the argument *beginning from the end of the list* and counting backward toward the first list element, and is sent out the right outlet. Any remaining list elements are sent out the left outlet of the object. Note: Lists are sent out the right outlet first.

mode group takes an additional number argument which specifies the size, in elements, of a list. A list received in the left inlet will be stored and the length of the list is compared to a number received in the right inlet or an argument to the **zl** object. If the list of items is longer than the number specified by the right inlet or argument, a list of items of the length specified by the number is sent out the left outlet. Any remaining symbols or list elements are stored.

mode iter takes an additional number argument which specifies the size, in elements, of a list. A symbol list of items received in the left inlet will be stored and sent out the left outlet as a series of lists consisting of the number of items specified by argument or by a number received in the right inlet. The final list output may be shorter than the specified number of items, depending on the stored contents of the **zl** object

mode join accepts a list in both inlets and sends a list out the left outlet which is the combination of both input lists.

mode len accepts a list in the left inlet and outputs number of elements in the list out the left outlet.

mode nth accepts a list in the left inlet and outputs the *nth* element of the list out the left outlet.

mode reg functions as a register that holds a list. A list received in the left inlet is sent out the left outlet immediately. A list received in the right inlet is stored. A bang sends the stored list out the left outlet.

mode rev accepts a list in its left inlet and sends the list out the left outlet in reverse order.

mode rot is used to rotate the contents of a list. An additional argument is used to specify the number of places a list item is to be rotated—positive numbers rotate the list to the right, and negative numbers rotate left. This value can also be specified as an input in the right inlet in this mode.

mode sect accepts a list in both inlets and sends a list out the left outlet that contains the elements common to both lists.

mode slice is used to divide a list into two lists. This mode takes an additional number argument which specifies the size, in elements, of a list. This value can also be specified as an input in the right inlet in this mode. A list received in the left inlet will be split into two lists—the first list contains the number of items specified by the argument, and is sent out the left outlet. Any remaining list elements are sent out the right outlet of the object. Note: Lists are sent out the right outlet first.

mode sub accepts a list in both inlets and sends the output position for each occurrence of right list in the left list out the left outlet.

mode union accepts a list in both inlets and sends a list out the left outlet that contains the contents of both input lists. If the left and right inlets contain any items in common, only one symbol will be output.

list In left inlet: The behavior of the **zl** object to lists received in the left inlet varies according to the mode of the object, as described above under the mode message.

list In right inlet: Some modes of **zl** accept a list in the right inlet and behave as follows:

Mode	*Behavior*
join	The list is joined with the list received in the left inlet, and output when a bang or list is sent to the left inlet.
reg	The list is stored, and sent out the left outlet when a bang is received by the left inlet.
sect	The list is stored; when a bang or list is sent to the left inlet, items common to both lists are sent out the left outlet.

sub	The list is stored; when a bang or list is sent to the left inlet, the output position for each occurrence of right list in the left list is sent out the left outlet.
union	The list is stored; when a bang or list is sent to the left inlet, a combination of both lists without repeating items common to both lists is sent out the left outlet.

bang In left inlet: Sends a list out the left or left and right outlets as follows:

Mode	*Behavior*
slice	Divides the currently stored list into two, outputs the last *N* items out the right outlet and any remaining items out the left outlet, where *N* is set by argument or a number received in the right inlet.
group	Outputs the most recently stored *N* items out the left outlet, where *N* is specified by argument or a number received in the right inlet.
iter	Outputs the most recently stored items out the left outlet in groups of a size specified by the argument or a number received in the right inlet.
join	Outputs the combination of the lists received in the left and right inlets out the left outlet.
mth	Outputs the *mth* element of the list designated by index. List numbering begins with 0.
nth	Outputs the *nth* element of the list designated by index.List numbering begins with 1.
reg	Outputs the currently stored list out the left outlet.
rev	Outputs the reverse of the currently stored list out the left outlet.
rot	Outputs the currently stored list with the contents rotated *N* places out the left outlet, where *N* is set by argument or a number received in the right inlet.
sect	Output the items in common to the lists received in the left and right inlets out the left outlet.
slice	Divides the currently stored list into two, outputs the first *N* items out the left outlet and any remaining items out the right outlet,

	where *N* is set by argument or a number received in the right inlet.
sub	Outputs the position for each occurrence of the list received in the right inlet in the list received in the left inlet. If an additional argument is used to specify a value which will be replace the number specified by the input value, the resulting list is sent out the right outlet of the **zl** object.
union	Output a list consisting of the items from both lists without repeating the items comment to both lists received in the left and right inlets out the left outlet.

int In right inlet: Some modes of **zl** accept an int in the right inlet and behave as follows:

Mode	*Behavior*
ecils	Specifies the number of list items beginning at the *end* of the input list to be sent out the right outlet of the **zl** object. Any remaining list elements beyond the number specified by this inlet are sent out the left outlet of the object.
group	Specifies a number of the most recently stored list items to be output.
iter	The currently stored contents of the **zl** object will be output as a series of lists consisting of the number of items specified by this value. The final list output may be shorter than the number, depending on the stored contents of the object.
mth	Specifies the order an element in the input list (using 0 as the index of the first element of the list) and outputs that element of the list.
nth	Specifies the order an element in the input list in numerical form (i.e., 1=the index of the first element of the list) and outputs that element of the list.
rot	Specifies the number of places to rotate the currently stored list. Positive values for rotate the list right, and negative values rotate left.
slice	Specifies the number of list items to be sent out the left outlet of the **zl** object. Any remaining list elements beyond the number specified by this inlet are sent out the right outlet of the object.

Arguments

symbol Optional. The words ecils, group, iter, join, len, mth, nth, reg, rev, rot, sect, slice, sub, or union are used as optional arguments to set the mode of the **zl** object. See the mode message above for descriptions of the different modes.

int Optional. For some modes of operation, an additional number may be used as an argument to specify the functionality of a given mode.

Mode	*Behavior*
ecils	Specifies the number of list items beginning at the *end* of the input list to be sent out the right outlet of the **zl** object. Any remaining list elements beyond the number specified by this inlet are sent out the left outlet of the object.
group	Specifies a number of the most recently stored list items to be output.
iter	The currently stored contents of the **zl** object will be output as a series of lists consisting of the number of items specified by this value. The final list output may be shorter than the number, depending on the stored contents of the object.
mth	Specifies the order of an element in the input list (using 0 as the index of the first element of the list) and outputs that element of the list.
nth	Specifies the order of an element in the input list in numerical form (i.e., 1=the index of the first element of the list) and outputs that element of the list.
rotate	Specifies the number of places to rotate the currently stored list. Positive values for rotate the list right, and negative values rotate left.
slice	Specifies the number of list items to be sent out the left outlet of the **zl** object. Any remaining list elements beyond the number specified by this value are sent out the right outlet of the object.
sub	The output position for the occurrence of the number specified by this value in the input list is sent out the left outlet of the object. An additional argument may be used to specify a value which will be replace the number specified by the input value. The resulting list is sent out the right outlet of the **zl** object.

Output

list Out left outlet:

In ecils mode, a list containing the number of elements specified by the number argument starting at the end of the list and counting toward the b.eginning.

In group mode, a list containing the number of elements specified by the number argument.

In iter mode, a number of lists having the number of elements specified by the number argument. The final list output may be shorter than the specified number of items, depending on the stored contents of the **zl** object

In join mode, a list containing all the elements of the lists received in both inlets.

In len mode, a number which corresponds to the number of list items.

In mth mode, the *mth* element of the list (where 0 is the index of the first element of the list).

In nth mode, the *nth* element of the list.

In reg mode, the input or the most recently stored input value received in the right inlet.

In rev mode, the input list in reverse order.

In rotate mode, the input list rotated to the right or left according to the positive or negative specified by the number argument.

In sect mode, a list containing all the elements common to the lists received in both inlets.

In slice mode, a list containing the number of elements specified by the number argument.

In union mode, a list containing the items from both lists without repeating items common to both lists. If the left and right inlets contain any items in common, only one symbol will be output.

list Out the right outlet:

In ecils mode, a list containing any list elements before the numbered element specified by the number argument.

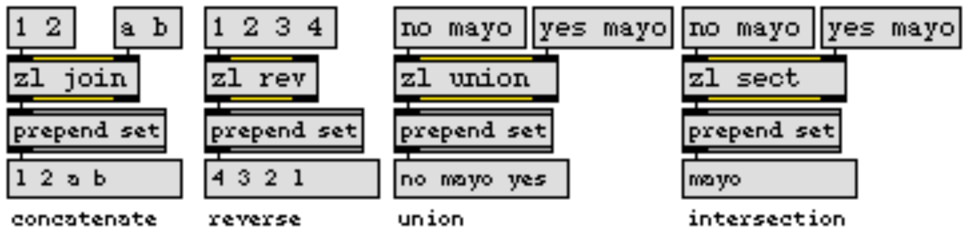
In mth mode, a list containing all list elements except for the list element specified by the number argument (where 0 is the index of the first element in the list).

In nth mode, a list containing all list elements except for the list element specified by the number argument (where 1 is the index of the first element in the list).

In slice mode, a list containing any list elements beyond the numbered element specified by the number argument.

In sub mode, the number of list elements specified by the number argument in the left input list is sent out the right outlet of the object. If an optional second argument is used to specify a value which will replace the number specified by the input value, the resulting list is sent out the right outlet of the **zl** object.

Examples



zl *is the Swiss Army Knife for lists*

See Also

fromsymbol	Transform a symbol into individual numbers or messages
maximum	Output the greatest in a list of numbers
minimum	Output the smallest in a list of numbers
pack	Combine numbers and symbols into a list
swap	Reverse the sequential order of two numbers
thresh	Combine numbers into a list, when received close together
tosymbol	Convert messages, numbers, or lists to a single symbol

zmap

Maps input range of values to output range

Input

int Converted to float.

float In left inlet: The incoming value is scaled according to the mapping provided by the arguments, or values received in the other inlets.

In second inlet: Sets the low input value. If the value is higher than the high input value, the two values are reversed to preserve the high-low relationship.

In third inlet: Sets the high input value. If the value is lower than the low input value, the two values are reversed to preserve the high-low relationship.

In fourth inlet: Sets the low output value. If the value is higher than the high output value, the two values are reversed to preserve the high-low relationship.

In right inlet: Sets the high output value. If the value is higher than the high output value, the two values are reversed to preserve the high-low relationship.

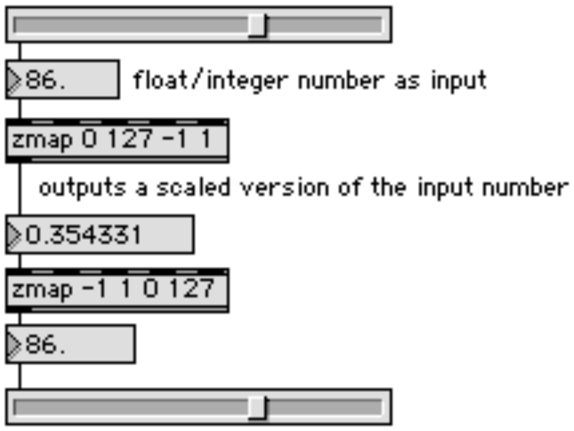
Arguments

int or float Optional. The first argument is the minimum input value, the second argument is the maximum input value. The third and fourth arguments are the minimum and maximum output values, respectively. If either of the low values is higher than the corresponding high value (or vice versa), the two values are reversed to preserve the high-low relationship.

Output

float When scale receives a value in its leftmost inlet, that value is scaled to the indicated output range of values.

Examples


86.
float/integer number as input
zmap 0 127 -1 1
outputs a scaled version of the input number
0.354331
zmap -1 1 0 127
86.

An example of how to map an integer slider into a useful range of floating-point values and back again

See Also

scale	Maps input to output range
expr	Evaluate a mathematical expression

Appendix

Some Max objects (such as **fpic**, **matrixctrl**, and **pictctrl**) will let you open and use an extended set of graphics files if you have QuickTime installed on your system. The following graphics file formats are currently supported:

MooV
sooV
TVex
MPG
MPEG
VfW
dvc!
FLI ',
GIFf
BINA
qmed
Cach
SWFL
RTSP
SDP
SwaT
SMI
JPEG
3DMF
MPGv
MPGx
BMP
8BPS
PNGf
PNG
qdgx
qtif
SGI
TPIC
TIFF
FLI

For an up-to-date list of graphics file formats supported by QuickTime, see

http://www.apple.com/quicktime/pdf/QuickTime_Pro_DS-b.pdf

See Also

fpic	Display a picture from a graphics file
lcd	Draw graphics in a patcher window
matrixcrtrl	Matrix-style switch control
pictctrl	Picture-based control
pictslider	Picture-based slider

Object Thesaurus

Objects listed by task keyword

Absolute to a relative path conversion	**relativepath**
Absolute value of an integer	**abs**, **expr**
Accelerate, control clock speed of Max timing objects	**setclock**
Action patch, receive events (messages) from a timeline	**ticmd**
Active sensing, MIDI system message	**midiin**, **midiout**, **rtin**
Add and/or multiply a series of numbers	**accum**, **expr**, **table**
Add two numbers together	**accum**, **expr**, **+**
Address elements in an array by index number	**counter**, **funbuff**, **offer**, **table**
ADSR envelope generator	**env**, **envi**
Afterpressure, polyphonic	**polyin**, **polyout**
Aftertouch (monophonic) MIDI message	**touchin**, **touchout**
Alert, display a text message	**dialog**, **lcd**, **umenu**, **message**, **pcontrol**, **print**
Alert, flash when an event occurs	**button**, **led**, **ubutton**
All notes off (MIDI Mode message)	**ctlin**, **ctlout**
And, true if both statements are true (logical intersection)	**expr**, **&&**
Animation of shapes or pictures	**frame**, **graphic**, **lcd**, **oval**, **pict**, **rect**, **ring**
Animation, control a laser videodisc player	**serial**, **vdp**
Animation, play a QuickTime movie	**imovie**, **movie**, **playbar**, **timeline**
Append items at the end or beginning of a message	**append**, **prepend**
Arc-cosine function	**acos**
Arc-sine function	**asin**
Arc-tangent function	**atan**
Arc-tangent function (two variables)	**atan2**
Arithmetic expression solving	**expr**, **+**, **-**, *****, **/**,**%**
Arithmetic operators	**acos**, **acosh**, **asin**, **asinh**, **atan**, **atan2**, **atanh**, **cosh**, **sin**, **sinh**, **sqrt**, **tan**, **tanh**
Array of arbitrary messages	**coll**, **umenu**
Array of numbers	**funbuff**, **histo**, **offer**, **table**
ASCII number for each character in a string	**spell**
ASCII number, convert to text character	**sprintf**
ASCII numbers, convert symbol to	**spell**
Ask for a file or folder	**opendialog**
Ask the user to enter information	**dialog**, **message**
Assistance, attach an assistance message to an inlet or outlet in a subpatch	**inlet**, **outlet**
Atoms of a list, break up into individual messages	**cycle**, **iter**, **message**, **spray**, **unpack**
Average a running stream of numbers	**mean**
Background panel	**panel**
Background, notify objects when patcher window is moved to background	**active**
bang a certain number of times as fast as possible	**uzi**
bang automatically when a patch is loaded	**loadbang**
bang messages, count	**counter**
bang repeatedly at a certain rate	**metro**
bang when a message is received or the mouse is clicked	**button**, **ubutton**
bang when a patcher window is closed	**closebang**
bang, cause all **loadbang** objects in a patcher window to resend	**thispatcher**
bang, send a single bang to different places in immediate succession	**bangbang**, **trigger**
bang, time elapsed between two bang messages	**timer**

Object Thesaurus

Objects listed by task keyword

Bend, report incoming MIDI pitchbend data .... **bendin**, **midiin xbendin**, **xbendin2**
Bend, transmit MIDI pitchbend messages .... **bendout**, **midiout**, **xbendout**, **xbendout2**
bang message traffic control .... **onebang**
Binary numbers, compare with bitwise "and" (intersection) and bitwise "or" (union) .... **expr**, **&**, **|**
Binary numbers, shift the bits of a number's binary representation to the left or right .... **<<**, **>>**
Binary, display numbers as .... **number box**
Bit-shift, shift the bits of the number's binary representation to the left or right .... **<<**, **>>**
Bitwise one's complement operation .... **expr**
Bitwise operators, bitwise "and" (intersection) and bitwise "or" (union) .... **expr**, **&**, **|**
Boolean logic operations .... **if**, **<**, **<=**, **==**, **!=**, **>=**, **>**, **&&**, **||**
Breakpoint line segment function generation and storage .... **env**, **envi**, **funbuff**, **line**
Breath control .... **ctlin**, **ctlout**
Broadcast a message to all instances of the same class in a patcher .... **universal**
Brownian motion simulator .... **drunk**
Button for user interface, sends a 1 or a 0 to start or stop processes .... **led**, **toggle**
Button for user interface, sends a bang .... **button**, **ubutton**
Button pasted over a picture or a comment .... **ubutton**
Button, picture-based .... **pictctrl**
C language expression solving .... **expr**, **if**
Capture and display a series of numbers .... **capture**, **print**, **table**, **text**
Cartesian to Polar coordinate conversion .... **cartopol**
CD audio control .... **cd**
Chance operations using pseudo-random numbers .... **drunk**, **expr**, **random**, **urn**
Characters in a string of text, convert to ASCII numbers .... **spell**
Check box user interface object .... **radiogroup**
Circle or oval, drawing in a graphic window .... **oval**, **ring**
Clock for reporting time elapsed .... **clocker**, **timer**
Clock speed of Max timing objects, control .... **setclock**
Clock, MIDI system message .... **midiin**, **midiout**, **rtin**
Close a patcher window automatically .... **pcontrol**, **thispatcher**
Closing a patcher window, send a bang when window is closed .... **closebang**
Collection of messages, store and recall .... **coll**, **umenu**
Color selection using a modal dialog .... **colorpicker**
Color swatch for RGB color selection and display .... **swatch**
Colored button area .... **panel**
Combinatorics, produce random orderings of a set .... **urn**
Commands, place your own commands in the menu bar .... **menubar**
Commands, send to a timeline from one of its own action patches .... **thistimeline**, **thistrack**, **tiout**
Commenting a patch .... **comment**
Compare a performance to a pre-recorded sequence in real time .... **follow**
Comparison of two numbers, less than, greater than, or equal to .... **if**, **<**, **<=**, **==**, **!=**, **>=**, **>**
Complement, bitwise one's complement operation .... **expr**
Compute x to the power of y .... **pow**
Computer keyboard events, detect .... **key**, **keyup**, **numkey**
Concatenate two messages .... **append**, **prepend**
Conditional statements .... **if**, **match**, **select**, **split**, **==**, **!=**, **<**, **>**, **<=**, **>=**

Object Thesaurus

Objects listed by task keyword

Connect patch cords to an inlet or outlet of a subpatch	**inlet**, **outlet**
Constrained random movement	**drunk**
Construct a list out of individual items	**append**, **pack**, **prepend**
Construct MIDI messages for transmission or recording	**midiformat**, **sxformat**
Continue, MIDI system message	**midiin**, **midiout**, **rtin**
Continuous controllers	**ctlin**, **ctlout**
Control a patcher window automatically from within itself	**thispatcher**
Control a timeline from one of its own action patches	**thistimeline**, **thistrack**, **tiout**
Control a videodisc player through the serial port	**vdp**
Control change messages	**ctlin**, **ctlout**
Control clock speed of Max timing objects	**setclock**
Control external (non-MIDI) device	**cd**, **serial**, **vdp**
Control points in a function	**env**, **envi**
Control strip for a QuickTime movie	**playbar**
Control, picture-based	**pictctrl**
Convert a number, list, or symbol to bang	**button**, **bangbang**, **trigger**
Convert an absolute to a relative path	**relativepath**
Convert ASCII numbers to text	**sprintf**
Convert numbers between decimal, hexadecimal, and binary forms	**number box**
Convert text to ASCII numbers	**spell**
Cosine function	**cos**
Count how many bang messages or numbers have been received	**counter**
Count the occurrences of numbers	**histo**
Count, send a series of numbers as fast as possible	**uzi**
Cumulative total of a series of numbers	**accum**, **expr**, **table**
Data structures, arbitrarily ordered array of arbitrary messages	**coll**, **umenu**
Date and time of day	**date**
Decimal numbers, store numbers with a fractional part	**float**, **number box**
Decrement the value of a user interface object	**IncDec**
Define a region for dragging and dropping a file	**dropfile**
Delay a bang for a specific amount of time	**delay**
Delay note-off messages until a specific event occurs	**sustain**
Delay one or more numbers for a specific amount of time	**pipe**, **thresh**
Delay, measure the time elapsed between two events	**borax**, **clocker**, **date**, **timer**
Delta time, report time interval between onsets of MIDI notes	**borax**, **timer**
Devices, drive external devices	**serial**, **vdp**
Devices, get a list of MIDI devices and ports currently available	**midiinfo**
Dial for sending numbers	**dial**
Difference between two numbers, obtain by subtracting	**expr**, **-**
Discrete values from a continuous stream of data	**speedlim**
Display numbers in decimal, hexadecimal, or binary form	**number box**
Display numerical data graphically	**dial**, **envi**, **hslider**, **kslider**, **multislider**, **number box**, **slider**, **table**, **uslider**
Display the zero/non-zero status of a number	**led**, **number box**, **toggle**
Distribute incoming numbers out individual outlets	**cycle**
Divide one number by another	**expr**, **/**

Divide two numbers, output the remainder .... **%**
Division object (inlets reversed) .... **!/**
Drag and drop .... **dropfile**
Draw a picture in a graphic window .... **graphic**, **pict**
Draw shapes in a graphic window .... **frame**, **graphic**, **oval**, **rect**, **ring**
Draw shapes in a patcher window .... **lcd**
Draw with the mouse .... **lcd**, **mousestate**
Duration, report length of MIDI notes .... **borax**
Duration, specify for transmitted MIDI notes .... **flush**, **makenote**, **midiflush**, **pipe**
Enable or disable MIDI objects in a patcher automatically .... **pcontrol**
End of a message, add items to .... **append**
Enter numerical data into a patcher from the computer keyboard .... **number box**, **numkey**
Enter text typed in by the user .... **dialog**, **message**
Envelope generator .... **env**, **envi**
Error messages, display text in a patcher window .... **dialog**, **lcd**, **umenu**, **message**, **pcontrol**
Error messages, print in the Max window .... **print**
Event number, assign to each MIDI note .... **borax**
Exclusive or, bitwise XOR operation .... **expr**
Export MIDI file .... **seq**
Expressions, solve mathematical .... **expr**, **+**, **-**, *****, **/**, **%**
External clock source, synchronize Max to .... **setclock**
Extra precision MIDI pitchbend messages .... **xbendin**, **xbendin2**, **xbendout**, **xbendout2**
Fader for displaying or generating numerical data .... **hslider**, **multislider**, **rslider**, **slider**, **uslider**
File modification date .... **filedate**
File menu, add your own items to .... **menubar**
File, import and export MIDI files .... **seq**
File, open any type of .... **filein**
Files, list the files in a specific folder .... **folder**
Film or video, synchronize Max to .... **setclock**
Filter a continuous stream of messages .... **speedlim**
Floating-point numbers, store numbers with a fractional part .... **float**, **number box**
Folder content listings .... **folder**
Follow a performance, comparing it to a pre-recorded sequence .... **follow**
Foreground, move a patcher window automatically to the front .... **thispatcher**
Foreground, notify objects when patcher window is brought to foreground .... **active**
Format MIDI messages for transmission or recording .... **midiformat**, **sxformat**
Format numbers, ASCII bytes, and symbols into a single message .... **sprintf**
Fourteen-bit precision MIDI pitchbend messages .... **xbendin**, **xbendout**, **xbendin2**
, **xbendout2**
Fraction, obtain by dividing one number by another .... **expr**, **/**
Fractions, store numbers with a fractional part .... **float**, **number box**, **pv**, **value**
Frequency, keep track of how many times a number has occurred .... **histo**
Full pathname to filename conversion .... **strippath**
Function generator .... **env**, **envi**, **funbuff**, **line**
Gate the flow of messages .... **gate**, **Ggate**
Generate numbers with the mouse ..... **dial**, **envi**, **hslider**, **imovie**, **kslider**, **lcd**, **mousestate**, **multislider**,

number box, **rslider**, **slider**, **table**, **uslider**
Get filename from an absolute pathname .... **strippath**
Global message-sending .... **float**, **forward**, **grab**, **int**, **message**, **receive**, **send**, **value**
Global variables .... **pv**, **value**
Graphic display of an array of numbers, editable with the mouse .... **multislider**, **table**
Graphic editor for arranging Max messages to be sent to specific objects at specific times . **timeline**
Graphics, draw a picture in a graphic window .... **pict**
Graphics, draw shapes in a graphic window .... **frame**, **graphic**, **oval**, **rect**, **ring**
Graphics, draw shapes in a patcher window .... **lcd**
Graphics, put a picture in a patcher window .... **fpic**
Greater than and less than comparisons, redirect numbers based on .... **split**
Greater than, find the greater of two numbers .... **expr**, **maximum**, **number box**, **peak**, **>**, **>=**
Greater than, report when all numbers in a list surpass specific thresholds .... **past**
Held MIDI notes, provide note-off messages for .... **borax**, **flush**, **makenote**, **midiflush**
Hexadecimal, display numbers as .... **number box**
Hierarchical on/off switch .... **decode**
Hint, pop-up menu .... **hint**
Histogram of how many times a number has occurred .... **histo**
Hold one or more numbers .... **float**, **funbuff**, **int**, **number box**, **offer**, **pv**, **table**, **value**
Hold the smallest in a series of numbers .... **trough**
Hyperbolic arc-cosine function .... **acosh**
Hyperbolic Arc-sine function .... **asinh**
Hyperbolic arc-tangent function .... **atanh**
Hyperbolic cosine function .... **cosh**
Hyperbolic sine function .... **sinh**
Hyperbolic tangent function .... **tanh**
If-then-else control structure .... **if**
Ignore certain messages .... **gate**, **Ggate**, **Gswitch**, **mousefilter**, **select**, **switch**
Import MIDI file .... **seq**
Incoming MIDI messages, parse .... **midiparse**, **xbendin**, **xnotein** also **bendin**, **ctlin**, **notein**, **pgmin**, **polyin**, **rtin**, **sysexin**, **touchin**
Increment the value of a user interface object .... **IncDec**
Index number, prepend to a number or a list .... **funnel**, **prepend**
Indexed list of numerical values .... **funbuff**, **offer**, **table**
Indicate the zero/non-zero status of a number .... **if**, **led**, **number box**, **togedge**, **toggle**, **==**,**!=**
Indicator flashes when a message is received .... **button**, **led**, **ubutton**
Information about current operating system and hardware .... **gestalt**
Initialize values automatically when a patch is loaded .... **loadbang**, **preset**
Inlet for a subpatch object .... **bpatcher**, **inlet**, **patcher**
Inlet, ignore messages in all inlets but one at a time .... **switch**
Input from the user, ask for .... **dialog**, **message**
Input received from MIDI devices, unaltered .... **midiin**
Integer number, store .... **funbuff**, **int**, **number box**, **offer**, **pv**, **table**, **value**
Intercept and redirect the output of an object .... **grab**
Inter-onset interval, measure the time elapsed between two events .... **borax**, **clocker**, **date**, **timer**
Interpolate between two numerical values .... **line**

Invert the color of a rectangular area of a patcher window over a picture or a comment	**ubutton**
Invisible button	**ubutton**
Invisible patcher, close	**thispatcher**
Invisible patcher, load	**pcontrol**
Items of a list, break up into individual messages	**cycle**, **iter**, **message**, **spray**, **unpack**
Keyboard style slider for displaying and generating numbers	**kslider**
Keyboard, detect computer keyboard events	**key**, **keyup**, **numkey**
Keydown event on computer keyboard, detect	**key**
Keyup event on computer keyboard, detect	**keyup**
Knob, picture-based	**pictctrl**
Label objects in a patcher window	**comment**, **umenu**
Laser disc player, control via the serial port	**serial**, **vdp**
Last (few) of a series of numbers are retained and sent out separate outlets	**bucket**
Less than and greater than comparisons, redirect numbers based on	**split**
Less than, find the lesser of two numbers	**expr**, **minimum**, **number box**, **trough**, **<**, **<=**
Limit the rate at which messages are sent	**speedlim**
List of indexed messages in a pop-up menu	**umenu**
List of numbers, detect a specific ordered set within	**match**
List processing	**zl**
List the files in a specific folder	**folder**
List, break up items into individual messages	**cycle**, **iter**, **message**, **spray**, **unpack**
List, combine separate items into	**append**, **pack**, **prepend**, **thresh**
List, evaluate a mathematical expression multiple times using lists of numbers as input	**vexpr**
List, graphically display and send out a list of number values	**multislider**
Lists, array of	**coll**, **umenu**
Load a patcher automatically	**pcontrol**
Local variable for any message, known only to a single patcher and its subpatches	**pv**
Local variable for storing a floating-point number (with a fractional part)	**float**, **number box**, **pv**
Local variable for storing an integer value	**int**, **number box**, **pv**
Logarithm of a number, solve for	**expr**
Loops, count repeated events	**counter**
Loops, repeated series of actions	**counter**, **metro**, **uzi**
Markov chain	**prob**
Masking, bitwise "and" (intersection) and bitwise "or" (union) operations	**expr**, **&**, **|**
Match incoming message to arguments, send a bang out a specific outlet if there is a match	**select**
Match the first item in a message, route the message accordingly	**route**
Mathematical expression solving	**expr**, **+**, **-**, *****, /, **%**
Matrix-style switch control	**matrixcrtrl**
Max search path information	**filepath**
Maximum and minimum limit for a range of numerical values, specify and display	**rslider**, **split**
Maximum, find the greater of two numbers	**expr**, **maximum**, **number box**, **peak**, **>**, **>=**
Maximum, find the maximum value of a group of numbers	**maximum**, **table**
Menu bar, customize or alter menus or menu items	**menubar**
Menu, pop-up menu in a patcher	**umenu**
Message symbol substitution	**substitute**
Messages, construct MIDI messages for transmission or recording	**midiformat**, **sxformat**

Messages, construct .... **append**, **message**, **pack**, **prepend**
Messages, send and display .... **umenu**, **message**
Messages, send remotely without patch cords ..... **float**, **forward**, **grab**, **int**, **message**, **pv**, **receive**, **send**, **value**
Messages, send with the menu bar .... **menubar**
Messages, type in and send in a locked patcher .... **dialog**, **message**
Metronome of timed events .... **clocker**, **metro**, **setclock**, **tempo**
MIDI, get a list of currently available devices and ports .... **midiinfo**
MIDI data, receive unaltered .... **midiin**
MIDI data, transmit byte by byte .... **midiout**
MIDI file, record, play, import, export, and save as text .... **seq**
MIDI Manager, synchronize Max to an external clock source .... **setclock**
MIDI messages, construct .... **midiformat**, **sxformat**, **midiout**
MIDI messages, parse .... **midiparse**, **xbendin**, **xnotein**
MIDI Mode messages .... **ctlin**, **ctlout**
MIDI note messages, receive incoming .... **midiin**, **notein**, **xnotein**
MIDI note messages, transmit .... **midiout**, **noteout**, **xnoteout**
MIDI note names, display numbers as .... **number box**
MIDI Real Time system messages .... **midiin**, **midiout**, **rtin**
MIDI Sample Dump, receive and transmit .... **midiin**, **midiout**, **sysexin**
MIDI, enable or disable MIDI objects in a patcher automatically .... **pcontrol**
Minimum and maximum limit for a range of numerical values, specify and display ..... **rslider**, **split**
Minimum, find the lesser of two numbers .... **expr**, **minimum**, **number box**, **trough**, **<**, **<=**
Minimum, find the minimum value of a group of numbers .... **minimum**, **table**
Minus, subtract one number from another .... **expr**, **-**
Modem communication, transmit and receive non-MIDI data .... **serial**
Modification date of a file .... **filedate**
Modulation wheel .... **ctlin**, **ctlout**
Modulus operation .... **expr**, **%**
Monitor size .... **screensize**
Monophonic aftertouch MIDI message .... **touchin**, **touchout**
Mouse button, pass numbers through only when the mouse button is up .... **mousefilter**
Mouse button, report status of .... **mousestate**
Mouse events, detect .... **imovie**, **lcd**, **mousefilter**, **mousestate**
Mouse location, report .... **imovie**, **lcd**, **mousestate**
Mouse, generate numbers with the mouse .... **dial**, **envi**, **hslider**, **imovie**, **kslider**, **lcd**, **mousestate**, **multislider**, **number box**, **rslider**, **slider**, **table**, **uslider**
Movie, play QuickTime .... **imovie**, **movie**, **playbar**, **timeline**
Multi-media programming .... **cd**, **graphic**, **lcd**, **imovie**, **movie**, **timeline**, **vdp**
Multiply and/or add a series of numbers .... **accum**, **expr**, **table**
Multiply two numbers .... **accum**, **expr**, *****
Multi-purpose list processor .... **zl**
Multi-track sequencer of MIDI messages or numbers .... **mtr**
Name user interface objects in a patcher window .... **comment**, **umenu**
Negative number, convert to positive number .... **abs**, **expr**

Nibble, examine selected bits of a number's binary representation	**&, /, |, <<, >>**
Noise, filtered noise generator	**drunk**
Noise, white noise generator	**expr**, **random**
Non-zero and zero numbers, notify when input changes from one to the other	**change**, **togedge**
Non-zero, test if a number or expression is	**change**, **if**, **led**, **select**, **split**, **togedge**, **toggle**, **==**, **!=**, **&&**, **||**
Not, bitwise one's complement operation	**expr**
Not, convert a non-zero number to 0 and vice versa	**expr**, **==**
Note data, receive incoming MIDI	**midiin**, **notein**, **xnotein**
Note information (duration, delta time, etc.) derived from MIDI note messages	**borax**
Note messages, transmit MIDI	**midiout**, **noteout**, **xnoteout**
Note-off messages with release velocity, detecting and formatting	**xnotein**, **xnoteout**
Note-off messages, hold until a specific event occurs	**sustain**
Note-off messages, supply for held or stuck MIDI note-ons	**bag**, **borax**, **flush**, **makenote**, **midiflush**
Note-off messages, suppress	**gate**, **stripnote**
Notes to yourself—or another user—in a patcher window	**comment**
Notify objects when patcher window is moved to foreground or background	**active**
Notify user when an event has occurred	**button**, **led**, **message**, **print**, **ubutton**
Number sequences, generate automatically	**counter**, **line**, **clocker**, **tempo**
Number, store	**float**, **funbuff**, **int**, **number box**, **offer**, **pv**, **table**, **value**
Numbers, convert between decimal, hexadecimal, and binary	**number box**
Numbers, generate with the mouse	**dial**, **envi**, **hslider**, **imovie**, **kslider**, **lcd**, **mousestate**, **multislider**, **number box**, **rslider**, **slider**, **table**, **uslider**
Nybble, examine selected bits of a number's binary representation	**&**, **|**, **<<**, **>>**
Object within an object	**bpatcher**, **patcher**
Occurrences, keep track of how many bang messages have occurred	**counter**
Occurrences, keep track of how many times a number has occurred	**histo**
Octal, display numbers in Roland octal format	**number box**
OMNI Mode On/Off (MIDI Mode message)	**ctlin**, **ctlout**
On/Off switch	**decode**, **toggle**
Open a dialog to ask for a file or folder	**opendialog**
Open a dialog to ask for a filename for saving	**savedialog**
Open a patcher automatically	**pcontrol**
Open patcher files automatically	**folder**, **pcontrol**
Operating system and hardware information	**gestalt**
Or, bitwise exclusive or (XOR) operation	**expr**
Or, true if one statement or the other is true (logical union)	**expr**, **||**
Order, send a number, bang, list, or symbol to different places in a specific order	**trigger**
Order, switch order of number messages	**fswap**, **message**, **swap**
Ordered set of numbers, detect	**match**
Outlet for a subpatch object	**bpatcher**, **outlet**, **patcher**
Outlet, send items of an incoming list out individual outlets	**spray**, **unpack**
Outlet, send messages out one of the outlets of a timeline object	**tiout**
Output MIDI data byte by byte	**midiout**
Output the monitor size	**screensize**
Oval or circle, drawing in a graphic window	**oval**, **ring**
Panel	**panel**

Panic, turn off held MIDI notes	**borax**, **ctlout**, **flush**, **makenote**, **midiflush**
Panning	**ctlin**, **ctlout**
Parameter change to a MIDI device	**ctlout**, **midiout**, **sxformat**
Parse incoming MIDI messages	**midiparse**, **xbendin**, **xnotein**, also **bendin**, **ctlin**, **notein**, **pgmin**, **polyin**, **rtin**, **sysexin**, **touchin**
Pass numbers through only when the mouse button is up	**mousefilter**
Patch change MIDI message	**pgmin**, **pgmout**
Patch cords, connect to an inlet or outlet of a subpatch	**inlet**, **outlet**
patcher within a patcher, the contents of which are visible	**bpatcher**
Peak hold, keep track of the greatest in a series of numbers	**peak**
Peek at values in other objects	**grab**
Permute a set in random order	**urn**
Picture, display a graphics file in a patcher window	**fpic**
Picture, display PICT file in a graphic window	**pict**
Picture-based control	**pictctrl**
Picture-based slider	**pictslider**
Pitchbend, report incoming MIDI pitchbend data	**bendin**, **midiin**, **xbendin**, **xbendin2**
Pitchbend, transmit MIDI pitchbend messages	**bendout**, **midiout**, **xbendout**, **xbendout2**
Play a QuickTime movie	**imovie**, **movie**, **playbar**, **timeline**
Play a sequence of Max messages to be sent to specific objects at specific times	**timeline**
Play sequences of past messages or numbers	**follow**, **mtr**, **seq**
Plus, add two numbers together	**accum**, **expr**, **+**
Polar to Cartesian coordinate conversion	**poltocar**
Poly mode, assign a unique voice number to each note being played	**borax**, **poly**
Polyphonic afterpressure	**polyin**
Pop-up menu in a patcher	**umenu**
Pop-up style hint text	**hint**
Portamento	**bendin**, **bendout**, **ctlin**, **ctlout**
Ports, get a list of MIDI devices and ports currently available	**midiinfo**
Positive version of a negative number	**abs**, **expr**
Postpone a bang	**delay**
Postpone a number or list	**pipe**, **thresh**
Postpone note-off messages until a specific event occurs	**sustain**
Potentiometer-like dial for sending numbers	**dial**
Power, one number to the power of another	**expr**
Prepend one message at the beginning of another	**prepend**
Preset, store and recall values for all user interface objects	**preset**
Print any message in the Max window	**print**
Probabilistic (stochastic) decision making	**drunk**, **prob**, **random**, **table**, **urn**
Probability, keep track of how many times a number has occurred	**histo**
Product of multiplying two numbers	**accum**, **expr**, *****
Program change MIDI message	**pgmin**, **pgmout**
Progress bar, graphic display	**hslider**, **slider**, **uslider**
Pseudo-random number generator	**drunk**, **expr**, **random**, **urn**
QuickDraw graphic commands, draw with	**lcd**
QuickTime movie, play	**imovie**, **movie**, **playbar**, **timeline**

Radio button user interface object .... **radiograph**
Ramp function, generate .... **line**, **timeline**
Random number generator .... **drunk**, **expr**, **random**, **urn**
Random walk .... **drunk**
Range of numerical values, specify and display minimum and maximum limits .... **rslider**, **split**
Rate at which messages are sent, limit .... **speedlim**
Rate, combine numbers into a single list if received faster than a certain speed .... **thresh**
Rate, control clock speed of Max timing objects .... **setclock**
Rate, send out beat numbers at a metronomic tempo .... **tempo**
Raw data from a file, read byte by byte .... **filein**
Raw MIDI data, receive and transmit .... **midiin**, **midiout**, **sysexin**
Read in a file of binary data .... **filein**
Real Time MIDI system messages .... **midiin**, **midiout**, **rtin**
Recall sequences of past messages or numbers .... **follow**, **mtr**, **seq**
Receive any message from any window .... **receive**
Receive MIDI data unaltered .... **midiin**
Receive only specific MIDI messages .... **bendin**, **ctlin**, **notein**, **pgmin**, **polyin**, **rtin**, **sysexin**, **touchin**
Recently received values are stored and recalled .... **bucket**
Record sequences of MIDI data or numbers .... **follow**, **mtr**, **seq**
Rectangle or square, drawing in a graphic window .... **frame**, **rect**
Redirect messages to a specific destination .... **gate**, **Ggate**, **grab**, **route**, **split**, **spray**, **unpack**
Release velocity, detecting and formatting note-off messages with .... **xnotein**, **xnoteout**
Remainder from dividing one number by another, modulus operation .... **expr**, **%**
Remote connection of objects, without patch cords .... **float**, **forward**, **grab**, **int**, **message**, **pv**, **receive**, **send**, **value**
Repeatedly send bang messages as fast as possible .... **uzi**
Repeatedly send output at a certain rate .... **clocker**, **metro**, **tempo**
Repetitions, count .... **counter**
Repetitions, suppress repeated numbers .... **change**
Report information about the current Max search path .... **filepath**
Report the modification date of a file .... **filedate**
Reports when application is suspended and resumed .... **suspend**
Reproduce a single bang to different places in immediate succession .... **bangbang**, **trigger**
Reverse the order of two number messages .... **fswap**, **message**, **swap**
RGB color selection and display swatch .... **swatch**
Ritardando, control clock speed of Max timing objects .... **setclock**
Rotate elements of a set of numbers, out successive outlets .... **bucket**, **cycle**
Route messages to a specific destination .... **gate**, **Ggate**, **route**, **split**, **spray**, **unpack**
Sampler, receive and transmit sound data via MIDI Sample Dump .... **midiin**, **midiout**, **sysexin**
Save As file dialog .... **savedialog**
Save, move to the foreground, or close a patcher window automatically .... **thispatcher**
Schedule a number or list to be sent at a future time .... **pipe**, **thresh**
Schedule an event for a future time .... **delay**
Score of Max messages to be sent to specific objects at specific times .... **timeline**
Score-following .... **follow**
Screen size .... **screensize**

Scroll through a list of messages	**umenu**
Search path information	**filepath**
Select a color using a modal dialog	**colorpicker**
Select specific values or symbols from incoming messages	**select**
Selector, route messages depending on the first item of each message	**route**
Send a message to all instances of the same class in a patcher	**universal**
Send a message to receive objects in any other window	**float**, **forward**, **grab**, **int**, **message**, **send**
Separate a list into its constituent elements	**cycle**, **iter**, **spray**, **unpack**
Sequence of numbers, detect a specific ordered set of numbers	**match**
Sequence of numbers, generate automatically	**counter**, **line**, **clocker**, **tempo**
Sequencer of Max messages to be sent to specific objects at specific times	**timeline**
Sequencer	**follow**, **mtr**, **seq**
Serial port, transmit and receive non-MIDI data	**serial**, **spell**
Series of numbers, break a list up into individual messages	**cycle**, **iter**, **message**, **spray**, **unpack**
Series of numbers, combine into a single list	**thresh**
Set (of fixed order and size) of integers; output all whenever one is modified	**bondo**
Set values automatically when a patch is loaded	**loadbang**, **preset**
Set, produce a random ordering of a set	**urn**
Set, store an unordered set of numbers	**bag**
Shift sequential input from one outlet to another	**bucket**, **cycle**
Simultaneity, send a series of bang messages or numbers in a single tick of Max's clock	**uzi**
Sine function	**sin**
Sine, cosine, tangent, and other trigonometric functions	**expr**
Slider to display or generate numerical data	**hslider**, **kslider**, **multislider**, **rslider**, **slider**, **uslider**
Slider, picture-based	**pictslider**
SMPTE time code, synchronize to an external source via MIDI Manager	**setclock**
Snapshot, store and recall instantaneous values of all user interface objects	**preset**
Sound sample data, receive and transmit via MIDI Sample Dump	**midiin**, **midiout**, **sysexin**
Sound, play in a QuickTime movie	**imovie**, **movie**, **playbar**, **timeline**
Sparse array of numbers	**funbuff**, **offer**
Speed, combine numbers into a single list if received faster than a certain rate	**thresh**
Speed, limit the rate at which messages are sent	**speedlim**
Sprites, pictures and geometric shapes	**frame**, **graphic**, **lcd**, **oval**, **pict**, **rect**, **ring**
Square or rectangle, drawing in a graphic window	**frame**, **rect**
Square root of a number	**sqrt**
Start a process by sending the bang message	**button**, **loadbang**, **ubutton**
Start activity automatically when a patch is loaded	**loadbang**
Start, MIDI system message	**midiin**, **midiout**, **rtin**
Steal voices, turn off old notes if too many new ones arrive	**poly**
Stochastic (probabilistic) decision making	**drunk**, **prob**, **random**, **table**, **urn**
Stop or alter the flow of messages	**gate**, **Ggate**, **Gswitch**, **speedlim**, **switch**
Stop, MIDI system message	**midiin**, **midiout**, **rtin**
Store a fixed-size set of integers; output all whenever one element is modified	**bondo**
Store a series of numbers in order in an editable window	**capture**, **table**, **text**
Store an unordered set of numbers	**bag**
Store and recall recently received values	**bucket**, **table**, **text**

Store and recall values of all user interface objects at a certain moment	**preset**
Store one or more numbers	**float**, **funbuff**, **int**, **number box**, **offer**, **pv**, **table**, **value**
Store, recall, and automatically delete x,y pairs of numbers	**offer**
String of text combining numbers, ASCII bytes, and symbols into a single message	**sprintf**
Stuck MIDI notes, avoid or turn off	**borax**, **flush**, **makenote**, **midiflush**
Subpatch in a box, visible from the patcher that contains it	**bpatcher**
Subpatch object (subroutine)	**bpatcher**, **patcher**
Subpatch object, create an inlet or outlet in	**inlet**, **outlet**
Substitute a symbol for another symbol in a message	**substitute**
Subtract one number from another	**expr**, **-**
Subtraction object (inlets reversed)	**!-**
Sum of a set of numbers	**accum**, **expr**, **table**
Sum of two numbers	**accum**, **expr**, **+**
Suppress note-off messages	**stripnote**
Suppress the flow of certain messages	**gate**, **Ggate**, **Gswitch**, **mousefilter**, **select**, **switch**
Sustain notes by holding note-off messages until a specific event occurs	**sustain**
Sustain pedal	**ctlin**, **ctlout**, **sustain**
Switch a process on and off	**led**, **togedge**, **toggle**, **ubutton**
Switch control matrix	**matrixcrtrl**
Switch the flow of messages on or off	**gate**, **Ggate**, **Gswitch**, **switch**, **toggle**
Symbol to message conversion	**fromsymbol**
Synchronize asynchronously arriving inputs, send them out together	**buddy**
Synchronize Max to an external clock source	**setclock**
System exclusive messages, construct and transmit	**midiout**, **sxformat**
System exclusive messages, receive	**midiin**, **sysexin**
System Reset, MIDI system message	**midiin**, **midiout**, **rtin**
Tag messages with a unique index number	**borax**, **funnel**, **poly**
Tangent function	**tan**
Tempo, control clock speed of Max timing objects	**setclock**
Tempo, send out beat numbers at a metronomic tempo	**tempo**
Test the equality of two numbers	**change**, **if**, **match**, **select**, **==**, **!=**
Test the zero/non-zero status of a number or expression	**change**, **if**, **led**, **match**, **select**, **split**, **togedge**, **toggle**, **==**, **!=**, **&&**, **||**
Test whether one number is greater than another	**maximum**, **number box**, **peak**, **>**, **>=**
Test whether one number is less than another	**minimum**, **number box**, **trough**, **<**, **<=**
Text file, load, play, and save a MIDI file as plain text	**seq**
Text file, open and save	**text**
Text input by the user, obtain	**dialog**, **message**
Text, convert to ASCII numbers	**spell**
Text, display automatically in a patcher	**dialog**, **lcd**, **umenu**, **message**, **pcontrol**, **sprintf**
Text, display in a patcher	**comment**, **fpic**, **message**
Text, format numbers, ASCII bytes, and symbols into a single message	**sprintf**
Text, print any message in the Max window	**print**
Threshold, report when numbers surpass	**past**
Timbre change on a MIDI synthesizer	**pgmout**
Time code, receive from an external source	**setclock**

Time elapsed between events, check	**clocker**, **date**, **timer**
Time of day and date	**date**
Timeline of Max messages to be sent to specific objects at specific times	**timeline**
Timeline, control a timeline track from its own action patch	**thistrack**
Timeline, control from one of its own action patches	**thistimeline**, **thistrack**, **tiout**
Timeline, receive events (messages) from	**ticmd**
Timeline, report the current time of	**thistimeline**
Timeline, send messages out one of the outlets of a **timeline** object	**tiout**
Times, multiply two numbers	**accum**, **expr**, *****
Toggle a process on and off	**led**, **togedge**, **toggle**, **ubutton**
Track, control a timeline track from its own action patch	**thistrack**
Track, record and play back a multi-track sequence of messages or numbers	**mtr**
Traffic control for bang messages	**onebang**
Transform a symbol into individual numbers or messages	**fromsymbol**
Transition probabilities, Markov chain	**prob**
Transmit a specific type of MIDI message	**bendout**, **ctlout**, **noteout**, **pgmout**, **polyout**, **touchout**
Transmit MIDI data byte by byte	**midiout**
Trap and redirect the output of an object	**grab**
Trap computer keyboard events	**key**, **keyup**, **numkey**
Trap mouse events	**imovie**, **lcd**, **mousestate**
Trap occurrences of a specific ordered set of numbers	**match**
Trap occurrences of specific numbers	**follow**, **match**, **route**, **select**, **==**
Trap occurrences of specific symbols	**route**, **select**
Trigger a process by sending the bang message	**button**, **loadbang**, **ubutton**
Trigger events automatically when a patch is loaded	**loadbang**
Trigger events based on notes played by the user	**follow**, **match**, **route**, **select**, **==**
Trigonometric functions	**expr**
True/false testing	**if**, **led**, **match**, **select**, **split**, **togedge**, **toggle**, **==**, **!=**, **<**, **>**, **<=**, **>=**
Type numerical data into a patcher	**number box**, **numkey**
Type text into a locked patcher	**dialog**, **message**
Variable for storing a floating-point number (with a fractional part)	**float**, **number box**, **pv**, **value**
Variable for storing an integer value	**int**, **number box**, **pv**, **value**
Variable for storing any message	**pv**, **value**
Variable that is private to a single patcher and its subpatches	**pv**
Vector math, evaluate an expression multiple times using lists of numbers as input	**vexpr**
Velocity of incoming MIDI note-on messages, obtain	**midiparse**, **notein**
Velocity, detecting and formatting note-off messages with release velocity	**xnotein**, **xnoteout**
Video or film, synchronize Max to	**setclock**
Videodisc player	**vdp**
Videodisc player, control via the serial port	**serial**, **vdp**
Virtual connection of objects, without patch cords	**float**, **forward**, **grab**, **int**, **message**, **pv**, **receive**, **send**, **value**
Voice number, assign a unique number to each note being played	**borax**, **poly**
Voice stealing, turn off old notes if too many new ones arrive	**poly**
Volume control MIDI message	**ctlin**, **ctlout**
Volume control of a QuickTime movie	**playbar**

Wait before allowing a number to pass..**pipe**, **speedlim**, **thresh**
Wait before doing something...**delay**
Wait for input in both inlets, then send out both numbers .... **buddy**
Weighted probabilities... **drunk**, **expr**, **random**, **table**, **urn**
Window being closed sends a bang ....**closebang**
Window for displaying graphic shapes and pictures ....**graphic**
Window moving to foreground or background sends 1 or 0...**active**
Window on a subpatch seen within the patcher that contains that subpatch .... **bpatcher**
Window, enable or disable MIDI automatically .... **pcontrol**
Window, open and close automatically...**pcontrol**, **thispatcher**
Windows, communicate between **float**, **forward**, **grab**, **inlet**, **int**,**message**, **outlet**, **receive**, **send**, **value**
XOR, bitwise "exclusive or" operation ....**expr**
Zero and non-zero numbers, notify when input changes from one to the other.....**change**, **togedge**
Zero, test if a number or expression is equal to..**change**, **if**, **led**, **select**, **split**, **togedge**, **toggle**, **==**, **!=**, **&&**, **||**

Index

Index

Index

Index

Index