

# Pure Data (PD) Object Reference Glossary

## 1. Introduction

The following is a list of Pure Data (PD) objects and edited excerpts of their provided documentation. Objects included are found in PD-extended 0.40.3, distributed at <http://puredata.info/downloads>. Not all objects are represented. Documentation excerpts from Miller Puckette, Michal Seta, Krzysztof Czaja, and Dave Sabine. Additional documentation, commentary, and editing by Christopher Ariza.

The included style guidelines and notes reflect the needs of an introductory course in PD programming. All users should consult internal PD documentation for complete information.

The on-line version of this document is available at the following URL: <http://www.flexatone.net/docs/pdg>.

The PDF version of this document is available at the following URL: <http://www.flexatone.net/docs/pdg.pdf>.

This document was last updated on Fri Feb 19 15:35:36 2010.

## 2. Event Objects

### 2.1. [!=]

Evaluate two numbers; output 1 if not equal, else 0.

### 2.2. [%]

Modulus.

### 2.3. [&]

Bitwise and.

### 2.4. [&&]

Logical and.

### 2.5. [\*]

Multiplier.

### 2.6. [+]

Addition.

### 2.7. [-]

Subtraction.

### 2.8. [/]

Division.

Note: it is always better to use multiplication by fraction for scaling downward.

### 2.9. [<]

Evaluate two numbers; output 1 if the left inlet value is less than the right inlet value, else 0.

### 2.10. [<<]

Left shift.

### 2.11. [<=]

Evaluate two numbers; output 1 if the left inlet value is less than or equal to the right inlet value, else 0.

### 2.12. [==]

Evaluate two numbers; output 1 if equal, else 0.

### 2.13. [>]

Evaluate two numbers; output 1 if the left inlet value is greater than the right inlet value, else 0.

### 2.14. [>=]

Evaluate two numbers; output 1 if the left inlet value is greater than or equal to the right inlet value, else 0.

### 2.15. [>>]

Right shift.

## 2.16. [abs]

Absolute value.

## 2.17. [append]

Add item to a list. [append] maintains a pointer to a scalar, or else an empty pointer to the head of a list. You may set the pointer using the leftmost inlet. The creation arguments specify the template of a new scalar to append, and the names of the fields (there should be at least one) you will wish to initialize. To append an object, send a number to the leftmost inlet. [append]'s pointer is updated to point to the new scalar, and the new pointer is also output.

## 2.18. [atan]

No documentation.

## 2.19. [atan2]

No documentation.

## 2.20. [bendin]

Read incoming pitch bend values from MIDI.

## 2.21. [clip]

Force a number between a range. Minimum and maximum can be assigned as creation arguments.

## 2.22. [cos]

No documentation.

## 2.23. [counter]

Count the number of bangs received. Can be used to create a list of increasing numbers that changes direction and can reset, jump to minimum, and jump to maximum. Arguments: [counter max], [counter min max], [counter dir min max]

## 2.24. [ctlin]

Read incoming Control Change message from MIDI.

## 2.25. [dbtopow]

No documentation.

## 2.26. [dbtorms]

The [dbtorms] and [rmstodb] objects convert from decibels to linear ("RMS") amplitude, so that 100 dB corresponds to an "RMS" of 1. Zero amplitude (strictly speaking, minus infinity dB) is clipped to zero dB, and zero dB, which should correspond to 0.0001 in "RMS", is instead rounded down to zero.

## 2.27. [dbtorms~]

No documentation.

## 2.28. [delay], [del]

Bang after a timed delay. The [delay] object sends a bang to its outlet after a delay in milliseconds specified by its right inlet or its creation argument. Only bang messages are delayed.

[delay]'s left inlet accepts a number, or one of two messages: "bang" or "stop". If a number is sent to its inlet, [delay] will set the delay time equal to that number and schedule the outgoing "bang". The "stop" method will inform [delay] to cancel its scheduled output.

[delay] accepts only one "bang" at a time. It cannot process multiple delays. In other words, sending a "bang" to a [delay] which is already set will reschedule its output, canceling the old one.

## 2.29. [div]

Takes a number in its left inlet and will divide that number by either the creation argument or the number given at its left inlet and will produce the result without a remainder. If no creation argument is given, then the default value is 1

## 2.30. [expr]

Evaluate arbitrary expressions.

See also: [expr~] (Section 3.15)

### 2.31. [float], [f]

Providing a float in the right inlet will store the value and not provide output. A bang or float in the left inlet sends output immediately.

See also: [int] (Section 2.35), [zl reg] (Section 2.76)

### 2.32. [ftom]

Frequency to midi pitch conversion

### 2.33. [gate]

Route one input to one or more outputs based on a number. The number of outlets can be configured with a creation argument. The leftmost inlet controls which outlet is active; outlet selection number counts from 1, where 0 is none. The rightmost inlet is the data to be gated.

### 2.34. [inlet]

Within an abstraction, provide an inlet for numbers or symbols.

Note: for signals use [inlet~]

See also: [outlet] (Section 2.49)

### 2.35. [int], [i]

Providing a number in the right inlet will store the value and not provide output. A bang or number in the left inlet sends output immediately. Can also be used to convert a float to an integer value.

See also: [float] (Section 2.31), [zl reg] (Section 2.76)

### 2.36. [key]

No documentation.

### 2.37. [keyname]

No documentation.

### 2.38. [keyup]

No documentation.

### 2.39. [line]

Ramp generator. [line]'s left inlet defines the "target" value. The right inlet defines the "time" value. The "target, time" pair of numbers inform [line] to produce a numeric "ramp" from its current value (whatever that might be at any given moment) to the new value within the allotted time which is defined at the right inlet.

See also: [line~] (Section 3.19), [vline~] (Section 3.48)

### 2.40. [loadbang]

Send "bang" automatically when patch loads. The inlet is inactive and serves no purpose.

### 2.41. [log]

Natural logarithm.

### 2.42. [max]

Returns the greater of the two numbers passed to its inlets. For example, if the creation argument (or right inlet) is equal to 10, and you send 9 to the left inlet then the object will return 10. If you pass it an 11, then object returns 11.

See also: [min] (Section 2.44)

### 2.43. [metro]

Sends a series of bangs at a constant rate.

### 2.44. [min]

Returns the lesser of the two numbers passed to its inlets. For example, if the creation argument (or right inlet) is equal to 10, and you send 9 to the left inlet then the object will return 9. If you pass it an 11, then object returns 10.

See also: [max] (Section 2.42)

### 2.45. [mod]

Takes a number in its left inlet and will divide that number by either the creation argument or the number given at its left inlet and will produce the value of the remainder at its outlet. If no creation argument is given, then the default value is 1.

## 2.46. [moses]

Moses takes numbers and outputs them at left if they're less than a control value, and at right if they're greater or equal to it. The creation argument initializes the control value (10 in this example) and the right inlet changes it.

## 2.47. [mtof]

MIDI pitch to frequency conversion.

See also: [mtof~] (Section 3.24)

## 2.48. [notein]

Read incoming stream of MIDI notes. The [notein] object reads incoming MIDI notes and reports their note number, velocity and channel number. Without the argument it reads from all MIDI channels.

## 2.49. [outlet]

Within an abstraction, provide an outlet for numbers or symbols.

Note: for signals use [outlet~]

See also: [inlet] (Section 2.34)

## 2.50. [pack]

The pack object takes a series of inputs and then outputs a concatenated list. The number of creation arguments determines the number of inlets while the type of creation arguments determines the types of messages that [pack] should expect to receive at each inlet - although with some peculiarities described below.

Any message to the first inlet will force [pack] to output its package - its list of values. A bang to the first inlet will force [pack] to output the current values without resetting any of them.

See also: [unpack] (Section 2.68)

## 2.51. [pgmin]

Read incoming Program Change message from MIDI.

## 2.52. [pipe]

Delay a message. Delay time is specified in milliseconds.

See also: [delay] (Section 2.28)

## 2.53. [pow]

Exponentiate a number. The object returns the value at the left inlet to the power of the right inlet where the left inlet is the base and the right inlet is the exponent. For example: 2 to the power of 2 = 4 (i.e. 2 Squared)

## 2.54. [powtodb]

No documentation.

## 2.55. [prepend]

Add left inlet message in front of construction argument message.

See also: [zl join] (Section 2.73)

## 2.56. [random]

Generates random numbers between 0 and N-1, where N is the creation argument or the value of the right inlet.

## 2.57. [rmstodb]

The [dbtorms] and [rmstodb] objects convert from decibels to linear ("RMS") amplitude, so that 100 dB corresponds to an "RMS" of 1 Zero amplitude (strictly speaking, minus infinity dB) is clipped to zero dB, and zero dB, which should correspond to 0.0001 in "RMS", is instead rounded down to zero.

## 2.58. [route]

Route messages according to their first element. Route can be given exact messages to match or data types. Route checks the first element of a message against each of its creation arguments, which may be numbers or symbols (but not a mixture of the two unless the data types are defined explicitly), then sends the messages through the appropriate outlets. If a match is found, and the message contains only ONE element, then a bang is sent out the corresponding outlet. If a match is found, and the message contains multiple elements (a list), then all the list elements except the first element is sent out the corresponding outlet. Differs from [select] in that the complete message (not just a bang) is passed for lists.

See also: [select] (Section 2.59)

## 2.59. [select], [sel]

In its simplest form [select] checks its input against the constant "6" (which is defined by the creation argument). If they match, the first outlet gives "bang"; otherwise the input is simply sent through to the second outlet.

[select] can also be used to match symbols like the example in the upper-right of this patch. It important to note that the FIRST creation argument indicates to the [select] object which data type to expect. If your first creation argument is a symbol, like "dog", then the object will test only symbols and numbers will be ignored!

See also: [route] (Section 2.58)

## 2.60. [sin]

No documentation.

## 2.61. [sqrt]

Square-root.

## 2.62. [stripnote]

Stripnote takes note-off (zero-velocity) messages out of a stream of MIDI-style note message and passes the others through unchanged.

## 2.63. [tabread]

Read from a named array by providing an index.

See also: [tabwrite] (Section 2.64)

## 2.64. [tabwrite]

Write to a named array by providing value and index pairs.

See also: [tabread] (Section 2.63)

## 2.65. [tan]

No documentation.

## 2.66. [timer]

Measures elapsed time between right inlet and left inlet bangs in milliseconds.

## 2.67. [trigger], [t]

Sequence messages in right-to-left order and convert data. Can be abbreviated as a f, float as f, bang as b, symbol as s, list as l, anything as a.

Whenever going from one outlet to multiple inlets with a message, use [trigger] to make explicit where messages are going.

## 2.68. [unpack]

Combine several atoms into one message. The pack object takes a series of inputs and then outputs a concatenated list. The number of creation arguments determines the number of inlets while the type of creation arguments determines the types of messages that [pack] should expect to receive at each inlet.

See also: [pack] (Section 2.50)

## 2.69. [until]

After receiving a number or a bang in the left inlet, [until] will continue to produce bangs until either the provided number is reached or a bang is received in the right inlet. Providing a number in the left inlet makes [until] function identical to [uzi] in Max.

## 2.70. [zl ecils]

Split a list provided in the left inlet at an index value provided in the right inlet. The remainder of the list is provided out the right outlet.

Note: index values start at 1 and count from the right of the list.

## 2.71. [zl group]

A list, of a size specified in the creation argument or by a value provided to the right inlet, is output after being received in the left inlet. Values can be provided to the left inlet either a single list or as series of values.

## 2.72. [zl iter]

Given a list in the left inlet, output lists of a size specified in the creation argument or by a value provided to the right inlet.

### 2.73. [zl join]

Join two lists; output is provided after list in left inlet is received.

### 2.74. [zl len]

Return the length of the list from the left outlet.

### 2.75. [zl nth]

Given an index value provided in the right inlet and a list in the left inlet, output the value at the specified index.

Note: index values start at 1. If the index is greater than the length of the list, no output is returned.

### 2.76. [zl reg]

Store a list in the right inlet; bang it out later.

### 2.77. [zl rev]

Reverse a list.

### 2.78. [zl rotate], [zl rot]

Rotate a list provided in the left inlet by the number provided in the right inlet.

### 2.79. [zl sect]

Output the elements that are common to both input lists.

### 2.80. [zl slice]

Split a list provided in the left inlet at an index value provided in the right inlet. The remainder of the list is provided out the left outlet.

Note: index values start at 1 and count from the left of the list.

### 2.81. [zl union]

Returns a combination of both lists.

### 2.82. [||]

Bitwise or.

### 2.83. [|||]

Logical or.

## 3. Signal Objects

### 3.1. [\*~]

Signal multiplier (scaling).

Note: rightmost inlet will not accept a signal input if an object creation argument is provided.

### 3.2. [+~]

Signal addition (mixing).

### 3.3. [-~]

Signal subtraction.

### 3.4. [/~]

Signal division.

Note: it is always better to use multiplication by a floating-point value than division.

### 3.5. [block~]

Block size and on/off control for DSP.

Only one object in any window; acts on entire window and sub-windows.

### 3.6. [bpf~]

Signal-controlled bandpass filter.

Note: signals are not permitted to control Q or center frequency.

See also: [vcf~] (Section 3.46)

### 3.7. [catch~]

No documentation.

See also: [throw~] (Section 3.45)

### 3.8. [clip~]

Force a signal between a minimum and a maximum value. When the input signal is beyond the range, the limit value is returned.

See also: [max~] (Section 3.21), [min~] (Section 3.22)

### 3.9. [dac~]

Real-time audio output for Pd. Accept arguments (numbers) which indicate which audio channels are to be used by Pd. By default, this object is stereo and communicate on audio channels 1 and 2 (left and right respectively).

### 3.10. [delay~]

Delay a signal in the left inlet by a number of samples as set in the right inlet. Multiply the delay time by the sampling rate to set the delay time in seconds.

See also: [delread~] (Section 3.11), [delwrite~] (Section 3.12), [vd~] (Section 3.47)

### 3.11. [delread~]

Delay a signal. One or more independent [delread~] objects can read a delay line from one named [delwrite~] object. The delay line must be named as a construction argument. A float input will set the delay time in milliseconds. Creation order matters: for non-recursive algorithms, create [delread~] after [delwrite~]. Similar to [tapout~] in Max/MSP.

See also: [delwrite~] (Section 3.12), [vd~] (Section 3.47), [delay~] (Section 3.10)

### 3.12. [delwrite~]

Delay a signal. Provide a signal as input to add to a delay line. One or more [delread~] or [vd~] objects can read from the delay line created with the [delwrite~] object. Creation order matters: for non-recursive algorithms, create [delread~] after [delwrite~]. Similar to [tapin~] in Max/MSP.

See also: [delread~] (Section 3.11), [vd~] (Section 3.47), [delay~] (Section 3.10)

### 3.13. [ead~]

Exponential attack decay envelope. First inlet triggers the envelope. Second inlet (or first construction argument) sets attack time. Third inlet (or second construction argument) set decay time.

See also: [vline~] (Section 3.48), [line] (Section 2.39), [line~] (Section 3.19)

### 3.14. [env~]

Envelope follower. The [env~] object takes a signal and outputs its RMS amplitude in dB (with 1 normalized to 100 dB.) Output is bounded below by zero. The analysis is "Hanning" (raised cosine) windowed.

### 3.15. [expr~]

Evaluate signals as mathematical expressions.

See also: [expr] (Section 2.30)

### 3.16. [ftom~]

Frequency to midi pitch conversion

### 3.17. [hip~]

One-pole high pass filter. Rolloff is fixed.

Note: signals are not permitted to control cutoff frequency.

### 3.18. [inlet~]

No documentation.

See also: [outlet~] (Section 3.28)

### 3.19. [line~]

Audio ramp generator. The [line~] object generates linear ramps whose levels and timing are determined by messages you send it. The messages may be a single target value (causing the output to jump to the target) or a target and a time in milliseconds (to start a new ramp).

See also: [vline~] (Section 3.48), [line] (Section 2.39)

### 3.20. [lop~]

One-pole low pass filter. Rolloff is fixed.

Note: signals are not permitted to control cutoff frequency.

Applications: A [lop~ 30] (or similar low frequency) is an excellent way to smooth a control signal from [line] into an audio signal suitable for use as an envelope.

See also: [moog~] (Section 3.23), [onepole~] (Section 3.26)

### 3.21. [max~]

Return the larger of two signals.

See also: [clip~] (Section 3.8), [min~] (Section 3.22)

### 3.22. [min~]

Return the smaller of two signals.

See also: [clip~] (Section 3.8), [max~] (Section 3.21)

### 3.23. [moog~]

Signal-controlled resonant lowpass filter.

See also: [lop~] (Section 3.20), [onepole~] (Section 3.26)

### 3.24. [mtof~]

MIDI pitch to frequency conversion.

See also: [mtof] (Section 2.47)

### 3.25. [noise~]

Uniformly distributed white noise; the output is between -1 and 1.

See also: [pink~] (Section 3.30)

### 3.26. [onepole~]

One pole lowpass filter with signal-controlled cutoff frequency.

See also: [lop~] (Section 3.20), [moog~] (Section 3.23)

### 3.27. [osc~]

Sine wave oscillator; output is between -1 and 1.

Applications: can be converted to a square wave with [expr~ \$1v > .5]

### 3.28. [outlet~]

No documentation.

See also: [inlet~] (Section 3.18)

### 3.29. [phasor~]

Sawtooth oscillator. Ramps from 0 to 1; can be considered a Sawtooth waveform between 0 and 1.

Applications: Can be converted to a square wave with [expr~ \$v1 > .5]. Pulse width modulation is available with a dynamic comparison using [expr~ \$v1 > \$v2]. Can be used to drive [tabread4~] and similar objects. A low frequency [phasor~] into a [bp~] can produce a sharp click.

### 3.30. [pink~]

Pink noise generator.

See also: [noise~] (Section 3.25)

### 3.31. [pow~]

Exponentiate a signal.

### 3.32. [readsf~]

Reading and playing a one or more channel AIFF or WAVE audio file. Similar to [sfplay~] in Max/MSP.

### 3.33. [receive~]

No documentation.

See also: [send~] (Section 3.36)

### 3.34. [rmstodb~]

No documentation.

### 3.35. [samphold~]

The [samphold~] object samples its left input whenever its right input decreases in value (as a [phasor~] does each period, for example.) Both inputs are audio signals.

The “set” message sets the output value (which continues to be updated as normal afterward.) The “reset” message causes [samphold~] to act as if the specified value were the most recent value of the control input. Use this, for example, if you reset the incoming phasor but don't want the jump reflected in the output. Plain “reset” is equivalent to “reset infinity” which forces the next input to be sampled.

### 3.36. [send~]

No documentation.

See also: [receive~] (Section 3.33)

### 3.37. [sig~]

Convert control message to a signal.

Note: while some tilde object inlets will convert floats into signals, better to explicitly convert to a signal with [sig~].

### 3.38. [snapshot~]

Convert a signal to a number on demand. The [snapshot~] object takes a signal and converts it to a control value whenever it receives a bang in its left outlet. This object is particularly useful for monitoring outputs.

### 3.39. [switch~]

Can be used to turn DSP of a patch or sub-patch on or off.

Only one object in any window; acts on entire window and sub-windows.

### 3.40. [tabread4~]

A 4-point interpolating table lookup object.

This object permits reading smoothly from any array at the audio rate with 4-point polynomial interpolation. Indexes should start at index 1 to permit initial interpolation.

See also: [tabread4~] (Section 3.40)

### 3.41. [tabreceive~]

Read a block of a signal from an array continuously.

See also: [tabsend~] (Section 3.42), [tabwrite~] (Section 3.43)

### 3.42. [tabsend~]

Writes one block of a signal continuously to an array.

See also: [tabwrite~] (Section 3.43), [tabreceive~] (Section 3.41)

### 3.43. [tabwrite~]

Write a signal in an array. Tabwrite~ records an audio signal sequentially into an array. Sending it "bang" writes from beginning to end of the array. To avoid writing all the way to the end, you can send a "stop" message at an appropriate later time. The "start" message allows skipping a number of samples at the beginning. (Starting and stopping occur on block boundaries, typically multiples of 64 samples, in the input signal.)

See also: [tabwrite~] (Section 3.43)

### 3.44. [threshold~]

Outputs bangs when an input signal increases beyond or falls below two thresholds. For each threshold there is a debounce time, a time in milliseconds before re-triggering is available. Construction arguments are high threshold, high threshold debounce time, low threshold, low threshold debounce time.

### 3.45. [throw~]

No documentation.

See also: [catch~] (Section 3.7)

### 3.46. [vcf~]

Signal-controlled bandpass filter.

Note: signals are not permitted to control Q.

See also: [bpf~] (Section 3.6)

### 3.47. [vd~]

Delay a signal. One or more independent [vd~] objects can read a delay line from one named [delwrite~] object. The delay line must be named as a construction argument. A signal input will set the delay time in milliseconds. Similar to [tapout~] in Max/MSP.

See also: [delwrite~] (Section 3.12), [delay~] (Section 3.10), [delread~] (Section 3.11)

### 3.48. [vline~]

High-precision audio ramp generator. The [vline~] object, like [line~], generates linear ramps whose levels and timing are determined by messages you send it. The messages consist of a target value, a time interval (zero if not supplied), and an initial delay (also zero if not supplied.) Ramps may start and stop between audio samples, in which case the output is interpolated accordingly.

Any number of future ramps may be scheduled and [vline~] will remember them and execute them in order. They must be specified in increasing order of initial delay however, since a segment cancels all planned segments at any future time.

See also: [vline~] (Section 3.48)

### 3.49. [wrap~]

Remainder modulo 1. The [wrap~] object gives the difference between the input and the largest integer not exceeding it (for positive numbers this is the fractional part).

## 4. Objects by Category

### 4.1. Communications

[catch~] (Section 3.7), [inlet] (Section 2.34), [inlet~] (Section 3.18), [outlet] (Section 2.49), [outlet~] (Section 3.28), [receive~] (Section 3.33), [send~] (Section 3.36), [throw~] (Section 3.45)

### 4.2. Data Conversion

[dbtopow] (Section 2.25), [dbtorms] (Section 2.26), [dbtorms~] (Section 2.27), [ftom] (Section 2.32), [ftom~] (Section 3.16), [mtof] (Section 2.47), [mtof~] (Section 3.24), [powtodb] (Section 2.54), [rmstodb] (Section 2.57), [rmstodb~] (Section 3.34)

### 4.3. Digital Signal Processing

[block~] (Section 3.5), [dac~] (Section 3.9), [delay~] (Section 3.10), [delread~] (Section 3.11), [delwrite~] (Section 3.12), [switch~] (Section 3.39), [vd~] (Section 3.47)

### 4.4. Envelope Generators

[ead~] (Section 3.13), [line~] (Section 3.19), [vline~] (Section 3.48)

### 4.5. Filters

[bpf~] (Section 3.6), [hip~] (Section 3.17), [lop~] (Section 3.20), [moog~] (Section 3.23), [onepole~] (Section 3.26), [vcf~] (Section 3.46)

### 4.6. Signal Generators

[expr~] (Section 3.15), [line~] (Section 3.19), [noise~] (Section 3.25), [osc~] (Section 3.27), [phasor~] (Section 3.29), [pink~] (Section 3.30), [samphold~] (Section 3.35), [sig~] (Section 3.37), [tabread4~] (Section 3.40), [vline~] (Section 3.48)

### 4.7. Logical Operators

[&] (Section 2.3), [&&] (Section 2.4), [<<] (Section 2.10), [>>] (Section 2.15), [||] (Section 2.82), [|||] (Section 2.83)

### 4.8. Math

[%/] (Section 2.2), [\*] (Section 2.5), [\*~] (Section 3.1), [+ ] (Section 2.6), [+~] (Section 3.2), [-] (Section 2.7), [-~] (Section 3.3), [/] (Section 2.8), [/~] (Section 3.4), [abs] (Section 2.16), [atan] (Section 2.18), [atan2] (Section 2.19), [cos] (Section 2.22), [div] (Section 2.29), [expr] (Section 2.30), [expr~] (Section 3.15), [log] (Section 2.41), [max] (Section 2.42), [max~] (Section 3.21), [min] (Section 2.44), [min~] (Section 3.22), [mod] (Section 2.45), [moses] (Section 2.46), [pow] (Section 2.53), [pow~] (Section 3.31), [sin] (Section 2.60), [sqrt] (Section 2.61), [tan] (Section 2.65), [wrap~] (Section 3.49)

### 4.9. Relational Operators

[!=] (Section 2.1), [<] (Section 2.9), [<=] (Section 2.11), [=] (Section 2.12), [>] (Section 2.13), [>=] (Section 2.14), [div] (Section 2.29), [mod] (Section 2.45)

## 4.10. Time

[delay] (Section 2.28), [metro] (Section 2.43), [pipe] (Section 2.52)

## Bibliography

Kreidler, J. 2009. "Programming Electronic Music in Pd." Wolke Publishing House. Internet: <http://www.pd-tutorial.com>.

Puckette, M. 1997. "Pure Data." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 224-227.

Puckette, M. 2002. "Using Pd as a score language." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 184-187.

Puckette, M. 2003. "Theory and Techniques of Electronic Music." Internet: <http://www.crca.ucsd.edu/~msp/techniques.htm>.

Puckette, M. 2006. "Pd Documentation." Internet: [http://www.crca.ucsd.edu/~msp/Pd\\_documentation](http://www.crca.ucsd.edu/~msp/Pd_documentation).