

## *Max Objects from Norm Jaffe*



Norman Jaffe  
Vancouver, British Columbia,  
Canada

December 11, 2016

## Contents

Contents . . . . .	ii
List of Figures . . . . .	iv
Document History . . . . .	v
Foreword . . . . .	vii
<i>bqt</i> . . . . .	1
<i>caseShift</i> . . . . .	4
<i>changes</i> . . . . .	5
<i>compares</i> . . . . .	6
<i>dataType</i> . . . . .	7
<i>fidget</i> . . . . .	8
<i>fileLogger</i> . . . . .	10
<i>gcd</i> . . . . .	11
<i>gvp</i> . . . . .	12
<i>ldp1550</i> . . . . .	17
<i>listen</i> . . . . .	21
<i>listType</i> . . . . .	26
<i>map1d</i> . . . . .	27
<i>map2d</i> . . . . .	32
<i>map3d</i> . . . . .	37
<i>memory</i> . . . . .	42
<i>mtc</i> . . . . .	44
<i>mtcTrack</i> . . . . .	48
<i>notX</i> . . . . .	50
<i>pfsm</i> . . . . .	51
<i>queue</i> . . . . .	57
<i>rcx</i> . . . . .	59
<i>senseX</i> . . . . .	63
<i>serialX</i> . . . . .	65
<i>shotgun</i> . . . . .	68
<i>spaceball</i> . . . . .	69
<i>speak</i> . . . . .	72
<i>stack</i> . . . . .	75
<i>sysLogger</i> . . . . .	77
<i>tcpClient</i> . . . . .	78
<i>tcpLocate</i> . . . . .	82
<i>tcpMultiServer</i> . . . . .	83
<i>tcpServer</i> . . . . .	86
<i>udpPort</i> . . . . .	89
<i>Vabs</i> . . . . .	91
<i>Vassemble</i> . . . . .	92
<i>Vceiling</i> . . . . .	93
<i>Vcollect</i> . . . . .	94
<i>Vcos</i> . . . . .	96
<i>Vdecode</i> . . . . .	97

<i>Vdistance</i>	98
<i>Vdrop</i>	99
<i>Vencode</i>	100
<i>Vexp</i>	101
<i>Vfloor</i>	102
<i>Vinvert</i>	103
<i>Vjet</i>	104
<i>Vlength</i>	105
<i>Vlog</i>	106
<i>Vltrim</i>	107
<i>Vmean</i>	108
<i>Vnegate</i>	110
<i>Vreduce</i>	111
<i>Vreverse</i>	112
<i>Vrotate</i>	113
<i>Vround</i>	114
<i>Vrtrim</i>	115
<i>Vscan</i>	116
<i>Vsegment</i>	117
<i>Vsin</i>	118
<i>Vsplit</i>	119
<i>Vsqrt</i>	120
<i>Vtake</i>	121
<i>Vtokenize</i>	122
<i>Vtrim</i>	123
<i>Vtruncate</i>	124
<i>Vunspell</i>	125
<i>wqt</i>	126
<i>x10</i>	129
<i>x10units</i>	133
Appendix <i>Phidgets</i>	134
Index	139

## List of Figures

1	OS X: Connecting a <i>gvp</i> object to a <i>serial</i> object . . . . .	16
2	OS 9: Connecting a <i>gvp</i> object to a <i>serialX</i> object . . . . .	16
3	OS X: Connecting an <i>ldp1550</i> object to a <i>serial</i> object . . . . .	20
4	OS 9: Connecting an <i>ldp1550</i> object to a <i>serialX</i> object . . . . .	20
5	An example language file for a <i>listen</i> object . . . . .	23
6	Another example language file for a <i>listen</i> object . . . . .	24
7	Syntax diagram for language files . . . . .	25
8	An example map file for a <i>map1d</i> object . . . . .	30
9	Syntax diagram for 1-D map files . . . . .	31
10	An example map file for a <i>map2d</i> object . . . . .	35
11	Syntax diagram for 2-D map files . . . . .	36
12	An example map file for a <i>map3d</i> object . . . . .	40
13	Syntax diagram for 3-D map files . . . . .	41
14	OS X: Connecting an <i>mtc</i> object to a <i>serial</i> object . . . . .	46
15	OS 9: Connecting an <i>mtc</i> object to a <i>serialX</i> object . . . . .	46
16	State diagram for <i>mtc</i> objects . . . . .	47
17	An example state file for a <i>pfsm</i> object . . . . .	54
18	State transition diagram for the example state file . . . . .	55
19	Syntax diagram for state files . . . . .	56
20	OS X: Connecting a <i>spaceball</i> object to a <i>serial</i> object . . . . .	70
21	OS 9: Connecting a <i>spaceball</i> object to a <i>serialX</i> object . . . . .	71
22	State diagram for <i>tcpClient</i> objects . . . . .	81
23	State diagram for <i>tcpMultiServer</i> objects . . . . .	85
24	State diagram for <i>tcpServer</i> objects . . . . .	88
25	Definition of the output of <i>Vmean</i> . . . . .	109
26	OS X: Connecting an <i>x10</i> object to a <i>serial</i> object . . . . .	132
27	OS 9: Connecting an <i>x10</i> object to a <i>serialX</i> object . . . . .	132

## Document History

<b>January 2006</b>	added the object <i>senseX</i>
<b>November 2005</b>	updated the object <i>rcx</i> to support Mac OS X
<b>July 2005</b>	corrected a bug with multiple <i>tcpClient</i> , <i>tcpMultiServer</i> or <i>tcpServer</i> objects being loaded at the same time; added the objects <i>shotgun</i> , <i>udpPort</i> and <i>Vsplit</i>
<b>May 2005</b>	corrected voice-change bug in <i>speak</i>
<b>September 2004</b>	modified the <i>tcpClient</i> , <i>tcpMultiServer</i> and <i>tcpServer</i> objects to support ‘raw’ mode transfers
<b>July 2004</b>	modified the <i>rcx</i> object to support sending messages
<b>January 2004</b>	documented the plugins used with the <i>fidget</i> object
<b>December 2003</b>	added support for don’t care values in the <i>map1d</i> , <i>map2d</i> and <i>map3d</i> objects
<b>November 2003</b>	added the object <i>fidget</i> ; removed a potential race condition in the <i>gvp</i> , <i>ldp1550</i> , <i>mtc</i> , <i>spaceball</i> and <i>x10</i> objects; restored the <i>sysLogger</i> object to the OS X distribution
<b>June 2003</b>	completed conversion to Carbon; dropped the <i>serialX</i> object from the OS X distribution, as the functionality is provided in the current <i>serial</i> object; dropped the <i>rcx</i> and <i>sysLogger</i> objects from the OS X distribution due to problems with libraries used by them; added the objects <i>Vassemble</i> , <i>Vdecode</i> , <i>Vencode</i> , <i>Vltrim</i> , <i>Vrtrim</i> , <i>Vtokenize</i> , <i>Vtrim</i> and <i>Vunspell</i>
<b>March 2003</b>	began conversion to Carbon, for use with Mac OS X
<b>October 2002</b>	converted objects to CodeWarrior 8; corrected logic errors in the objects <i>map1d</i> , <i>map2d</i> , <i>map3d</i> , <i>tcpClient</i> and <i>tcpServer</i> ; added code to support forcing the state of the DTR signal in the <i>serialX</i> object
<b>September 2002</b>	modified the objects <i>tcpClient</i> , <i>tcpMultiServer</i> and <i>tcpServer</i> to accept a parameter of the number of receive buffers to use; improved the handling of fast messages for the objects <i>tcpClient</i> , <i>tcpMultiServer</i> and <i>tcpServer</i>
<b>August 2002</b>	added the object <i>Vcollect</i>
<b>June 2002</b>	converted objects to CodeWarrior 7, which no longer supports the Motorola MC68xxx processors, so all new objects will be PowerPC-only – the older objects will still be available for both (in a <i>Max 3.x</i> archive or a <i>Max 4.x</i> archive)
<b>April 2002</b>	added the object <i>rcx</i> ; improved the handling of non-standard list elements for all objects
<b>March 2002</b>	added the ‘ <b>kind</b> ’ command to the object <i>x10</i> ; modified the object <i>x10</i> to accept a parameter of the kind of X-10 controller attached; modified the objects <i>listType</i> , <i>pfsm</i> , <i>serialX</i> , <i>speak</i> , <i>tcpClient</i> , <i>tcpMultiServer</i> and <i>tcpServer</i> to properly handle the <i>Max</i> special characters; added the objects <i>fileLogger</i> and <i>sysLogger</i>
<b>February 2002</b>	began restructuring for <i>Max 4.x</i> ; modified internal resources so that <i>Max</i> collectives and standalone applications can be built

<b>January 2002</b>	added the ‘ <b>dtr</b> ’ command to the object <i>serialX</i> ; enhanced the output format for the objects <i>map1d</i> , <i>map2d</i> and <i>map3d</i>
<b>October 2001</b>	added an optional argument to the objects <i>Vceiling</i> , <i>Vfloor</i> , <i>Vround</i> and <i>Vtruncate</i> indicating the desired output format; modified the object <i>mtc</i> ’s ‘ <b>raw</b> ’ format to return all the data in a single vector rather than a series of rows; modified the object <i>serialX</i> to support external clocking
<b>August 2001</b>	began documenting compatibility with <i>Max 4.x</i>
<b>July 2001</b>	added the ‘ <b>self</b> ’ command to the objects <i>tcpClient</i> , <i>tcpMultiServer</i> and <i>tcpServer</i> ; added the ‘ <b>mode</b> ’ command to the object <i>mtc</i> ; added the object <i>spaceball</i>
<b>May 2001</b>	extended the format of the input file for the objects <i>map1d</i> , <i>map2d</i> and <i>map3d</i> to return offset/match data; added the objects <i>Vcos</i> , <i>Vexp</i> and <i>Vsin</i> ; added the ‘ <b>add</b> ’, ‘ <b>after</b> ’, ‘ <b>before</b> ’, ‘ <b>clear</b> ’, ‘ <b>count</b> ’, ‘ <b>delete</b> ’, ‘ <b>dump</b> ’, ‘ <b>get</b> ’, ‘ <b>replace</b> ’, ‘ <b>set</b> ’ and ‘ <b>show</b> ’ commands to the objects <i>map1d</i> , <i>map2d</i> and <i>map3d</i> ; added the ‘ <b>threshold</b> ’ command to the object <i>mtcTrack</i>
<b>April 2001</b>	improved the handling of the structural sections, such as the Table of Contents and modified macros that were impacted by generation of PDF; added the objects <i>listen</i> , <i>map3d</i> , <i>mtcTrack</i> , <i>queue</i> , <i>speak</i> , <i>Vabs</i> , <i>Vdistance</i> , <i>Vinvert</i> , <i>Vlog</i> , <i>Vmean</i> , <i>Vnegate</i> , <i>Vreverse</i> , <i>Vrotate</i> and <i>Vsqrt</i>
<b>March 2001</b>	improved page layout and simplified the use of $\text{\LaTeX 2}_{\epsilon}$ commands; replaced most of the Adobe Illustrator® graphics with MetaPost versions
<b>February 2001</b>	added cross-references and improved index and graphics inclusion
<b>January 2001</b>	converted this document to $\text{\LaTeX 2}_{\epsilon}$
<b>November 2000</b>	made a minor correction to the description of the ‘ <b>send</b> ’ command for the object <i>tcpMultiServer</i> ; clarified the ‘ <b>status</b> ’ commands for the objects <i>tcpClient</i> , <i>tcpMultiServer</i> and <i>tcpServer</i> ; added documentation for the TCP/IP message format to the object <i>tcpClient</i> ; modified the TCP/IP message format to remove extraneous fields and increase robustness (the resulting TCP/IP objects are not compatible with earlier versions of the same objects); added the objects <i>map1d</i> , <i>map2d</i> , <i>tcpLocate</i> , <i>Vceiling</i> , <i>Vfloor</i> , <i>Vreduce</i> , <i>Vround</i> , <i>Vscan</i> and <i>Vtruncate</i>
<b>October 2000</b>	added the objects <i>caseShift</i> , <i>listType</i> and <i>tcpMultiServer</i> ; added the ‘ <b>float</b> ’ command to the object <i>serialX</i> ; added the ‘ <b>status</b> ’ command to the objects <i>tcpClient</i> and <i>tcpServer</i> ; renamed the objects <i>drop</i> , <i>jet</i> , <i>length</i> , <i>segment</i> and <i>take</i> to <i>Vdrop</i> , <i>Vjet</i> , <i>Vlength</i> , <i>Vsegment</i> and <i>Vtake</i> , respectively; added the ‘ <b>float</b> ’ and ‘ <b>list</b> ’ commands to the object <i>notX</i> ; corrected the description of the arguments to the object <i>changes</i>
<b>September 2000</b>	first version of this document

## Foreword

This document is a description of the *Max* objects that I've written over the past several years, for myself and my friends, and will be a 'living' document—I intend to update it as new objects join the family. It is intended for *Max* programmers, and assumes a basic understanding of how *Max* works—I've made no attempt to explain standard *Max* objects or how to use my objects with standard *Max* objects, unless there are special considerations for use of my objects with standard *Max* objects. I don't provide examples, except in the form of online help files, as some of the objects are more easily understood from experimentation than exposition. This is not to say that the objects are difficult to use—I consider myself a lazy *Max* programmer and an experienced *C* programmer, so I've attempted to make the objects robust and responsive, with few idiosyncrasies. Some of the objects may show a *LISP* or an *APL* flavour, which is more a reflection on my languages of choice than an indication that *Max* has or doesn't have these elements. The objects were written using [Metrowerks CodeWarrior](#), [Apple Macintosh Programmer's Workshop](#) (MPW), [Apple ResEdit™](#) and the *Max* Software Development Kit from [Cycling '74](#). But before I get into the goodies, some preliminaries:

Copyright: © 2000 T. H. Schiphorst, N. Jaffe, K. Gregory and G. I. Gregson.

All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*Max* is a program written by Miller Puckette and David Zicarelli, © 1990–2004 [Cycling '74](#) / [ircam](#).

Metrowerks CodeWarrior copyright © 1993–2002 [Metrowerks Inc.](#) and its licensors. All rights reserved. Metrowerks, the Metrowerks logo, and CodeWarrior are registered trademarks of [Metrowerks Inc.](#), a Motorola company.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Distiller, Illustrator, Photoshop and PageMaker are trademarks of [Adobe Systems Incorporated](#).

Apple, Applescript, Mac, the Mac logo, Macintosh, MPW, QuickTime and ResEdit are trademarks of [Apple Computer, Inc.](#). QuickTime and the QuickTime logo are trademarks used under licence. IBM and PowerPC are registered trademarks of [International Business Machines Corporation](#).

dvips(k) copyright © 2002 [Radical Eye Software](#).

BBEdit Lite copyright © 1992–1999 [Bare Bones Software Incorporated](#).

Phidgets copyright © 2003 [Phidgets Incorporated](#).

UserLand Frontier copyright © 1992–1998 [UserLand Software, Incorporated](#).

Syslogd copyright © 1998–2000 [Brian Bergstrand](#).

LEGO, LEGO MINDSTORMS, Robotics Invention System and RCX are registered trademarks of [The LEGO Company](#).

VOODOO Server © 1999–2002 [uni software plus gmbh](#).

Stuffit Deluxe copyright © 1990–2001 [Allume Systems, Inc.](#)

AFPL Ghostscript copyright © 2002 [artofcode LLC, Benicia, CA](#).

GNU Make copyright © 1988, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 2000 [Free Software Foundation, Inc.](#)

CMaxTex copyright © 1992–2004 [T. Kiffe](#).

SubEthaEdit copyright © 2003–2004 [TheCodingMonkeys](#).

TeXShop copyright © 2001–2004 [Eugene Algorithms](#).

For each object, I present an image of the object, showing its inlets and outlets, along with the following text entries:

- 1) A description of the object.
- 2) The year that the object was created.
- 3) Whether there is an online help file for the object. Wherever possible, the help file provides a comprehensive set of examples of use of the object. For some objects, the help file cannot convey the full complexity of the object—especially for objects with nontrivial state. For some objects, their command language is too complex to be placed in a help file, without making the help file difficult to use.
- 4) What theme or category the object belongs to. I’ve defined the following groups of objects:
  - a) Device interface (*fidget*, *gvp*, *ldp1550*, *listen*, *mtc*, *rcx*, *serialX*, *spaceball*, *speak*, *x10*)
  - b) Miscellaneous (*caseShift*, *dataType*, *fileLogger*, *gcd*, *listType*, *mtcTrack*, *notX*, *shotgun*, *sysLogger*, *x10units*)
  - c) Programming aids (*changes*, *compares*, *map1d*, *map2d*, *map3d*, *memory*, *pfsm*, *queue*, *selectX??*, *stack*)
  - d) QuickTime™ (*bqt*, *wqt*)
  - e) TCP/IP (*tcpClient*, *tcpLocate*, *tcpMultiServer*, *tcpServer*, *udpPort*)
  - f) Vector manipulation (*Vabs*, *Vceiling*, *Vcollect*, *Vcos*, *Vdistance*, *Vdrop*, *Vexp*, *Vfloor*, *Vinvert*, *Vjet*, *Vlength*, *Vlog*, *Vmean*, *Vnegate*, *Vreduce*, *Vreverse*, *Vrotate*, *Vround*, *Vscan*, *Vsegment*, *Vsin*, *Vsplit*, *Vsqr*, *Vtake*, *Vtruncate*)
- 5) Which class list(s) the object appears in within *Max*.
- 6) A description of the arguments, if any, for the object. Any special values are identified, and default values are identified for optional arguments.
- 7) A description of each of the inlets for the object. The domain of values for the inlets are identified. If the values have a special format, it is described later.
- 8) A description of each of the outlets for the object. The range of values for the outlets are identified, where possible.
- 9) The names of other objects that are normally used with the object. An example is the *serialX* object, which is used with *x10* and *ldp1550* objects.
- 10) Whether the object is standalone or communicates with other objects. The TCP/IP objects *tcpClient*, *tcpMultiServer* and *tcpServer* work with each other; a *memory* object works with other *memory* objects that have the same tag.



- 11) Whether the object retains state. An object that doesn't retain state will respond to a given signal presented to an inlet in the same way each time that the signal appears; an object that does retain state, such as a *stack* or *memory* object, may respond differently each time that the same signal appears.
- 12) Which versions of Max can use the object (*Max 3.x*, *Max 4.x* or *Max 4.5+*) and which operating systems are supported (OS 9 or OS X).
- 13) Whether the object can be used only on older Macintoshes (68K-only), Power Macintoshes only (PPC-only) or either (Fat). Note that the objects were all built to work with *Max 3.5* or newer.
- 14) If the object uses a command language, such as the inputs for the *memory* or *x10* objects, it is described in detail.
- 15) If the object uses an external file, such as *bqt* or *pfsm* objects, its format is described in detail.
- 16) Miscellaneous comments or anecdotes. This is whatever I felt needed to be mentioned, that didn't quite fit in one of the other categories.

For those objects, such as *gvp*, that require specific connections to other objects, a simple diagram of the non-obvious connections is provided. Where it would assist in understanding, I provide a state diagram or a syntax chart.

I'd like to thank the following people for inspiring me to write the objects described here, and for motivating me to actually document them:

My best friend, Thecla Schiphorst, who has helped me find my focus.

My friends Sang Mah, Ken Gregory and Grant Gregson, who've presented *Max* programming challenges to me on many occasions, and who've been my unwitting (but willing) guinea pigs for years.

The students of IA308, 2002, [Technical University of British Columbia](#) (now known as the Simon Fraser University Surrey campus), for inspiring me and invigorating me.

My friends Larry Wasik and Glen Taylor, who've helped me learn how to refine my code, and when refining is not necessary.

My friends Jerry Barenholtz, Tom Calvert, Ron Harrop and Doug Seeley for helping me find order and discipline within the chaos of software design and development.

My friends Ron McOuat and Chris Duncombe, who've shown me that a cool head can accomplish great things and overcome obstacles, both from people and computers.

My friends Torsten Belschner, Robb Lovell, Stock and Aadjan van der Helm for providing valuable feedback and ideas about these objects.

The cover page photograph was originally made by Larry Wasik. It's the card deck for the program CONTK, along with a (partial) listing of the source code. CONTK, if you're interested, was a process control program written in *FORTRAN IV* for the [IBM 1800](#) Process Control Computer, to control a Kamyr digester. If those terms aren't familiar to you, don't be surprised—I included the photograph as an attempt at humour, reflecting on my early days of programming.

Norm Jaffe,  
Vancouver, British Columbia, Canada  
December 11, 2016



An [OpenDragon](#) production.



Powered by Macintosh! Developed using (over the years) Macintosh II, Macintosh IIfx, Power Macintosh 9600, iBook, and Power Mac G4 machines.



Powered by CodeWarrior.

This document was created using [CMacTeX](#) 4.3, [SubEthaEdit](#) 2.1.1, BBEEdit Lite 6.1, AFPL Ghostscript 8.00, GNU Make version 3.79.1, Adobe® Illustrator® 10.0.3, AppleScript 1.9.3, Adobe® Photoshop® 7.0.1, [MetaPost](#) 0.641, [TeXShop](#) 1.35 and dvips(k) 5.92b on an Apple Power Mac G4.

***bqt***

[No image—*bqt* is a user interface object]

**Description of object:** *bqt* provides an interface to QuickTime™ movies, permitting control of playback rate and the section of the movie to be played. It provides more functionality than the standard *movie* player interface object

**Object created:** October 1998

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** QuickTime™

**Object class(es):** n/a

**Argument(s):** none

**Inlet(s):**

- 1 **list**, the command input

**Outlet(s):**

- 1 **integer**, the result of a command
- 2 **integer**, the duration of the current movie
- 3 **bang**, playing has stopped
- 4 **bang**, an error was detected

**Companion object(s):** yes, (optional) the standard *movie* play controller interface object can be attached to a *bqt* object.

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**active [0/1]**

Set the current movie active (**1**) or inactive (**0**).

**bang**

Start the current movie playing.

**begin**

Move to the beginning of the current movie and make it active.

**count**

Return the number of movies loaded.

**duration**

Return the length of the current movie.

**end**

Move to the end of the current movie and make it active.

**getrate**

Return the rate at which the current movie will be played.

**getvolume**

Return the audio level for the current movie.

**integer**

Move to the given frame number in the current movie.

**load [movie-name]**

Add the specified movie to the list of movies and make it the current movie. **movie-name** must be a symbol, not a number.

**mute [0/1]**

Change the audio level of the current movie, silencing it (**0**) or restoring the previous level (**1**).

**pause**

Stop the current movie.

**rate integer [integer]**

Set the rate at which the current movie will be played, using the ratio of the first number to the second, and start the movie playing. If only one number is given, or the second number is zero, assume that the second number has the value one.

**resume**

Continue playing the current movie after a '**pause**' or a '**stop**'.

**segment integer [integer]**

Set the portion of the current movie that will be played to the section from the first frame number to the second. If the first number is zero and the second number is zero or less, set the portion to be the whole movie. If the second number is negative, set the portion to be from the first frame number to the end of the movie. That is, '0 0' is the whole movie, as is '0 -1', while '15 -1' is the portion from frame 15 to the end.

**start**

Move to the beginning of the current movie, make it active and start it playing.

**stop**

Stop the current movie.

**time**

Return the current frame number of the current movie.

**unload [movie-name]**

If no movie is specified, remove the current movie from the list of movies. Otherwise, remove the specified movie from the list of movies. **movie-name** must be a symbol, not a number.

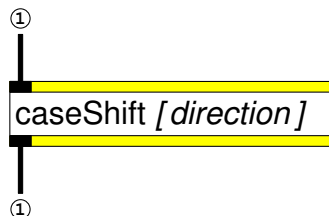
**volume [integer]**

Set the audio level of the current movie. The maximum level is 255; setting the level negative acts to mute the current movie, but the **'mute'** command can restore the audio level to the corresponding positive value.

**File format:** QuickTime™ movie

**Comments:** The *bqt* object was designed to address a critical weakness of the standard *movie* player interface object: there was no way to request only a section of the movie be played, even though QuickTime™ supports this ability. One feature of the standard *movie* player interface object was not retained—mouse motion over the *bqt* object is not detected.

## *caseShift*



**Description of object:** *caseShift* changes the case of each symbol in a list to either lower case or upper case.

**Object created:** October 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Messages

**Argument(s):**

*direction* (optional) symbol, the direction of the shift, either 'up' or 'down'. The default direction is 'up'.

**Inlet(s):**

1 anything, the symbols to be modified

**Outlet(s):**

1 anything, the symbols after case conversion

**Companion object(s):** none

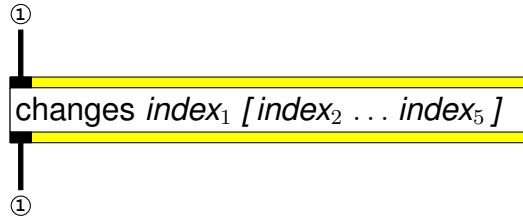
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## *changes*



**Description of object:** *changes* monitors an arbitrary list, watching for a change in specified elements. When a change is detected, the list seen in the inlet is sent to the outlet and it is remembered for comparison with subsequent lists.

**Object created:** October 1998

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Lists

**Argument(s):**

*index1* **integer**, the first list element index to monitor (indices start at 1; negative indices count from the end of the list)

*index2* **(optional) integer**, the second list element index to monitor

*index3* **(optional) integer**, the third list element index to monitor

*index4* **(optional) integer**, the fourth list element index to monitor

*index5* **(optional) integer**, the fifth list element index to monitor

**Inlet(s):**

**1 list**, the list to be monitored

**Outlet(s):**

**1 list**, the list matching input

**Companion object(s):** none

**Standalone:** yes

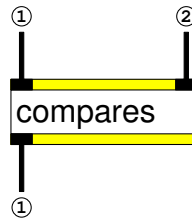
**Retains state:** yes, the elements of the previous input list

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat



## *compares*



**Description of object:** *compares* does a case-sensitive string comparison of two symbols.

**Object created:** April 1999

**Current version:** 1.0.3

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Arith/Logic/Bitwise

**Argument(s):** none

**Inlet(s):**

- 1 symbol**, the first string
- 2 symbol**, the second string

**Outlet(s):**

- 1 integer**, the comparison result ( $-1$  = second string greater,  $0$  = strings match,  $1$  = first string greater)

**Companion object(s):** none

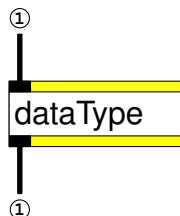
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## *data*Type



**Description of object:** *data*Type returns a numeric code corresponding to the value it receives.

**Object created:** October 1998

**Current version:** 1.0.3

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Lists

**Argument(s):** none

**Inlet(s):**

**1** anything, any value

**Outlet(s):**

**1** integer, the code for the input (unknown = 0, bang = 1, float = 2, integer = 3, list = 4, symbol = 5)

**Companion object(s):** none

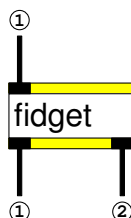
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## *fidget*



**Description of object:** *fidget* provides an interface to *Phidgets*® – physical widgets, available from [Phidgets Incorporated](#). The *fidget* object uses a folder named ‘Phidgets’, located in the same folder as the *Max* program, to provide a collection of ‘plugins’ for device-specific behaviours.

**Object created:** November 2003

**Current version:** 1.0.1

**Online help file:** no

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):** none

**Inlet(s):**

1 **list**, the command input

**Outlet(s):**

1 **list**, the command response

2 **integer**, the error code if an error was detected

**Companion object(s):** see *Phidgets* for the available ‘plugins’

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

**do deviceType serialNumber/\* [anything]**

Perform a device-specific operation for the specified phidget or all phidgets of the same type (if ‘\*’ is specified). The default behaviour is to do nothing.

**get deviceType serialNumber/\* [anything]**

Acquire data from the specified phidget or all phidgets of the same type (if '\*' is specified). The default behaviour is to interpret the argument as an element number and to return the value(s) from that element in the form of a list, with the first two elements of the list consisting of the 'deviceType' and the 'serialNumber'.

**listen [on/off]**

Start reporting phidget insertions and removals ('on') or stop reporting ('off'). If no argument is given, reporting is reversed.

**put deviceType serialNumber/\* [anything]**

Send data to the specified phidget or all phidgets of the same type (if '\*' is specified). The default behaviour is to interpret the argument as an element number followed by the value(s) to be sent to the element. There is no expected output from the 'put' command.

**report [deviceType [serialNumber/\* [element]]]**

If there are no arguments to the 'report' command, return a list of all devices, in the form of the symbol 'devices' followed by each device, as represented by its device type and serial number. If only the 'deviceType' is specified, return a list of all devices with matching device types, in the form of the symbol 'device' followed by the device type and the serial numbers of the matching devices. If the 'deviceType' and 'serialNumber' are specified, the elements of the matching device will be returned, in the form of a list with the symbol 'elements' followed by a triple for each element, consisting of the element 'cookie', the element type ('input-misc', 'input-button', 'input-axis', 'input-scancodes', 'output', 'feature' or 'collection') and the element size in bits. If '\*' is specified instead of 'serialNumber', 'element' is ignored and all phidgets of the same type are reported in the form of a series of lists with the symbol 'elements' followed by a triple for each element, consisting of the element 'cookie', the element type ('input-misc', 'input-button', 'input-axis', 'input-scancodes', 'output', 'feature' or 'collection') and the element size in bits. If all three arguments are present, the specified element is returned as a list starting with the symbol 'element' followed by the element type and size. Note that no output is returned if a match fails.

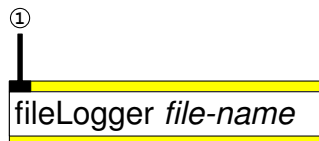
**structure deviceType serialNumber/\***

Report the hierarchical structure of the specified phidget or all phidgets of the same type (if '\*' is specified) to the *Max* window.

**Comments:**

*fidget* loads all available files from a folder named 'Phidgets', located in the same folder as the *Max* program, to provide a collection of 'plugins' for device-specific behaviours. Refer to *Phidgets* for a list of the currently available 'plugins'. Device serial numbers are prefixed with an underscore character ("\_") to guarantee their interpretation as symbols rather than numbers. Note that at most one *fidget* object can be loaded into *Max* due to the way that it interacts with the Macintosh device routines.

## *fileLogger*



**Description of object:** *fileLogger* writes its input to a standard file.

**Object created:** March 2002

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Miscellaneous

**Argument(s):**

*file-name* symbol, the file to be written to

**Inlet(s):**

**1** anything, the input to be processed

**Outlet(s):** none

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## *gcd*



**Description of object:** *gcd* calculates the greatest common divisor of two numbers.

**Object created:** October 1998

**Current version:** 1.0.3

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise

**Argument(s):** none

**Inlet(s):**

1 **bang/integer**, the first number to use

2 **integer**, the second number to use

**Outlet(s):**

1 **integer**, the greatest common divisor of the two numbers, or the previous result (if a '**bang**' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the previous number input

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

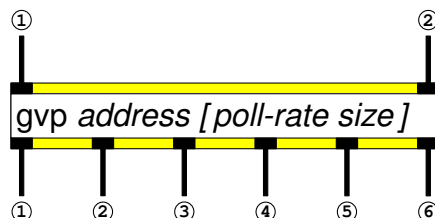
**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

Return the previous result, if any.

**Comments:** In mathematical terms:  $y \leftarrow \gcd(x_1, x_2)$ , where  $x_1$  and  $x_2$  are the inlet values and  $y$  is the outlet result.

***gvp*****Description of object:**

*gvp* is an interface to an ECHOLab DV7 video switcher, which utilizes the Grass Valley Protocol. It sends commands to a *serial* or a *serialX* object, which controls the serial port that the video switcher is attached to, and responds to data returned from the video switcher via the *serial* or *serialX* object.

**Object created:**

June 1998

**Current version:**

1.0.5

**Online help file:**

no

**Object theme:**

Device interface

**Object class(es):**

Devices

**Argument(s):**

**address integer**, the select address of the video switcher. The address must be even and less than 256.

**poll-rate (optional) integer**, the rate (in milliseconds) at which the companion *serial* or *serialX* object is polled via a sample request. The default rate is 100 milliseconds between sample requests.

**size (optional) integer**, the size of the command pool that is used to buffer commands to the video switcher. The default size is 50 and the maximum size is 200.

**Inlet(s):**

- 1 **list**, the command channel
- 2 **anything**, the output of the companion *serial* or *serialX* object

**Outlet(s):**

- 1 **bang**, the sequence has completed
- 2 **bang**, the command has completed
- 3 **bang**, the sample request to send to the companion *serial* or *serialX* object
- 4 **anything**, the data to send to the companion *serial* or *serialX* object
- 5 **bang**, the 'break' request to send to the companion *serial* or *serialX* object, via a message object
- 6 **bang**, an error was detected

**Companion object(s):**

works with *serialX* objects with OS 9, or *serial* objects with OS X

<b>Standalone:</b>	yes
<b>Retains state:</b>	yes
<b>Compatibility:</b>	<i>Max 3.x</i> and <i>Max 4.x</i> {OS 9 and OS X}
<b>Fat, PPC-only or 68k-only:</b>	Fat

**Command language syntax:****allstop**

Send the ‘allstop’ command to the video switcher.

**breakdone**

The ‘break’ sent to the *serial* or *serialX* object has completed. This command shouldn’t be sent under other circumstances.

**c bus-settings**

This is a shortcut for the ‘crosspoint bus-settings’ command.

**crosspoint bus-settings**

The argument **bus-settings** is a series of pairs of symbols and symbols or integers— if the list is of odd length, the value zero is added at the end. The first value is the name of the bus of the video switcher that is to be set (‘pgm’/‘program’, ‘preset’/‘preview’, ‘key’/‘insert’/‘k’/‘i’) and the second value is the source for the given bus (‘black’, ‘background’/‘b’ or an integer between 0 and 9, where ‘black’ is the same as ‘0’ and ‘background’ is the same as ‘9’). Each of the resulting pairs is sent to the video switcher to change the program, preset or key busses.

**!c level**

This is a shortcut for the ‘!clip level’ command.

**!clip level**

Set the clipping level to the given value, **level**, which is a floating point number that is between 0 and 100. The command is the equivalent to ‘dskanalogcontrol 8 level’.

**d control-values**

A shortcut for the ‘dskanalogcontrol control-values’ command.

**dskanalogcontrol control-values**

The argument **control-values** is a series of pairs of floating point numbers—if the list is of odd length, the value zero is added at the end. The first number in each pair is the control to be set and the second number in each pair is the value to set the control to. Each of the resulting pairs is sent to the video switcher to change the DSK analog control settings.

**e control-values**

This is a shortcut for the ‘effectsanalogcontrol control-values’ command.

**effectsanalogcontrol control-values**

The argument **control-values** is a series of pairs of floating point numbers—if the list is of odd length, the value zero is added at the end. The first number in each pair is the control to be set and the second number in each pair is the value to set the control to. Each of the resulting pairs is sent to the video switcher to change the effects analog control settings.

**!e position**

This is a shortcut for the ‘!effects position’ command.



**!effects position**

Set the ‘effects’ control position of the video switcher to the given value, **position**, which is a floating point value between 0 and 100. The command is equivalent to ‘**effectsanalogcontrol 21 position**’.

**endsequence**

Add the pseudo-command ‘endsequence’ to the buffer of commands to be sent to the video switcher. The pseudo-command ‘endsequence’ is not actually sent to the video switcher, but results in a ‘**bang**’ being sent out the first outlet to signal that a sequence of commands has been completed.

**!j horizontal vertical**

This is a shortcut for the ‘**!joystick horizontal vertical**’ command.

**!joystick horizontal vertical**

Set the coordinates of the joystick on the video switcher. The given values, **horizontal** and **vertical**, are floating point numbers between 0 and 100. The command is equivalent to ‘**effectsanalogcontrol 18 horizontal 17 vertical**’.

**learn-emem [register]**

The video switcher settings will be stored in the given **register** of the video switcher, where **register** is an integer between 0 and 15. If no value for **register** is given, zero is assumed.

**m keyOrBackground1 [keyOrBackground2]**

This is a shortcut for the ‘**transitionmode keyOrBackground1 [keyOrBackground2]**’ command.

**off buttons**

The argument **buttons** is a list of numbers, which are interpreted as the indices of the buttons and switches of the video switcher. The indices are between 0 and 80, but not all values correspond to actual buttons or switches. Each of the matching buttons and switches on the video switcher will have their state set to ‘off’. If an erroneous index appears, the remaining indices will be ignored.

**on buttons**

The argument **buttons** is a list of numbers, which are interpreted as the indices of the buttons and switches of the video switcher. The indices are between 0 and 80, but not all values correspond to actual buttons or switches. Each of the matching buttons and switches on the video switcher will have their state set to ‘on’. If an erroneous index appears, the remaining indices will be ignored.

**p buttons**

This is a shortcut for the ‘**push buttons**’ command.

**push buttons**

The argument **buttons** is a list of numbers, which are interpreted as the indices of the buttons and switches of the video switcher. The indices are between 0 and 80, but not all values correspond to actual buttons or switches. Each of the matching buttons and switches on the video switcher will have their state flipped (from ‘off’ to ‘on’ or vice-versa). If an erroneous index appears, the remaining indices will be ignored.

**r transition [rate [keyOrBackgroundOrDolt1 [keyOrBackgroundOrDolt2 [keyOrBackgroundOrDolt3]]]]**

This is a shortcut for the ‘**transitionrate transition [rate [keyOrBackgroundOrDolt1 [keyOrBackgroundOrDolt2 [keyOrBackgroundOrDolt3]]]]**’ command.

**recall-emem [register]**

The previously stored video switcher settings will be retrieved from the given **register** of the video switcher, where **register** is an integer between 0 and 15. If no value for **register** is given, zero is assumed.

**!t position**

This is a shortcut for the **!take position** command.

**!take position**

Set the 'take' control position of the video switcher to the given value, **position**, which is a floating point value between 0 and 100. The command is equivalent to **effectsanalogcontrol 11 position**.

**transitionmode keyOrBackground1 [keyOrBackground2]**

Set the active transition mode of the video switcher to either key ('key' or 'k') or background ('background' or 'b') or both.

**transitionrate transition [rate [keyOrBackgroundOrDolt1 [keyOrBackgroundOrDolt2 [keyOrBackgroundOrDolt3]]]]**

Set the transition rate for the given **transition** ('auto'/'a', 'dsk'/'d' or 'f2b'/'f', where 'f2b' represents fade-to-black) to the given value, **rate**, which is an integer between 0 and 1000. The transition rate affects either the key ('key' or 'k') or background ('background' or 'b') transitions. If the symbol 'doit' or 'd' appears, the transition is triggered as well. If no value for **rate** is given, zero is assumed.

**v [on/off]**

This is a shortcut for the **verbose [on/off]** command.

**verbose [on/off]**

Communication tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**w [index]**

This is a shortcut for the **wipepattern [index]** command.

**wipepattern [index]**

Set the wipe pattern of the video switcher to **index**, which is one of the following integer values: 0, 1, 3, 4, 10, 11, 20, 23, 30 or 33. If no value for **index** is given, zero is assumed.

**x**

This is a shortcut for the **xreset** command.

**xreset**

Remove any pending commands for the video switcher and send a 'break' to the video switcher to force a device reset.

**Comments:**

Figure 1 shows how to connect a *gvp* object to a *serial* object. Figure 2 shows how to connect a *gvp* object to a *serialX* object.

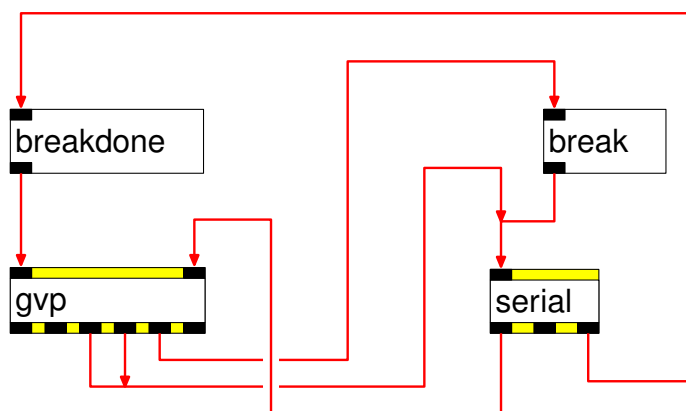


Figure 1: OS X: Connecting a *gvp* object to a *serial* object

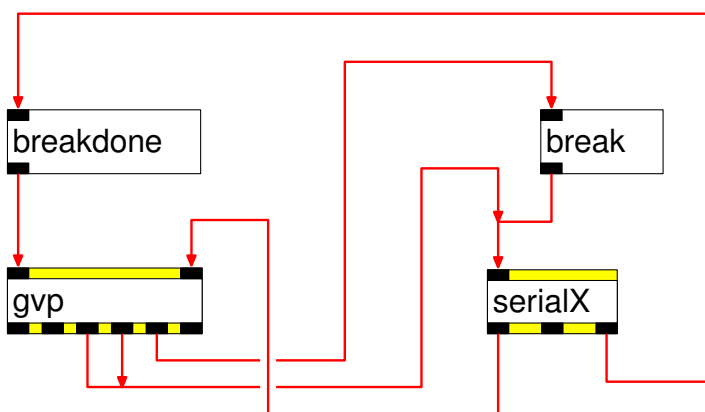
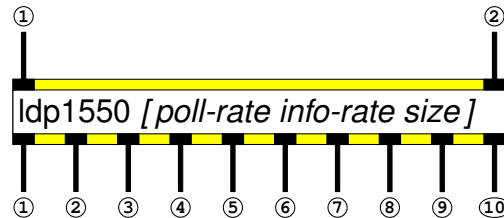


Figure 2: OS 9: Connecting a *gvp* object to a *serialX* object

## *ldp1550*



**Description of object:** *ldp1550* is an interface to the Sony Corporation LDP-1550 Laser Disk Player (LDP). It sends commands to a *serial* or a *serialX* object, which controls the serial port that the LDP is attached to, and responds to data returned from the LDP via the *serial* or *serialX* object.

**Object created:** September 1996

**Current version:** 1.0.6

**Online help file:** no

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):**

*poll-rate* **(optional) integer**, the rate (in milliseconds) at which the companion *serial* or *serialX* object is polled via a sample request. The default rate is 100 milliseconds between sample requests.

*info-rate* **(optional) integer**, the multiples of the poll-rate at which the laser disk player is sent a status request. The default rate is 10 times the poll-rate.

*size* **(optional) integer**, the size of the deferred command pool, where commands that can't be acted on immediately are held. The default size is 30 elements. Most commands consume several elements.

**Inlet(s):**

- 1 **list**, the command channel
- 2 **integer**, the output of the companion *serial* or *serialX* object

**Outlet(s):**

- 1 **integer**, the data to send to the companion *serial* or *serialX* object
- 2 **bang**, the sample request to send to the companion *serial* or *serialX* object
- 3 **integer**, the key mode status
- 4 **integer**, the command status
- 5 **bang**, a program stop code was detected
- 6 **bang**, the command has completed

- 7 **bang**, the command was accepted
- 8 **integer**, the chapter number
- 9 **integer**, the frame number
- 10 **bang**, an error was detected

**Companion object(s):** works with *serialX* objects with OS 9, or *serial* objects with OS X

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**allinfo**

Send commands to the laser disk player requesting the current chapter, the current frame number, the key mode status and the command status.

**chapter**

Send a command to the laser disk player requesting the current chapter.

**continue**

Send a 'continue' command to the laser disk player.

**frame**

Send a command to the laser disk player requesting the current frame number.

**memory**

Send a 'memory' command to the laser disk player.

**mode new-mode**

Send a command to the laser disk player requesting the mode, as given by **new-mode**, be set to 'chapter' or 'frame'.

**msearch**

Send an 'msearch' command to the laser disk player.

**picture on/off**

Send a command to the laser disk player, turning the video signal 'on' or 'off'.

**play [speedAndMode [stepRate]]**

Send a command to the laser disk player to begin playing the laser disk, at the given speed and mode ('fwd', 'fast', 'slow', 'scan', 'step', 'rev', 'rev-fast', 'rev-slow', 'rev-scan' or 'rev-step', where 'rev' indicates the reverse direction and 'fwd' indicates the forward direction and normal speed) and stepping rate, **stepRate**, which is an integer between 0 and 255. The default speed is forward, the default mode is normal and the default stepping rate is 0.

**playtill position [speed [stepRate]]**

Send a command to the laser disk player to begin playing the laser disk, at the given **speed** ('fwd', 'fast', 'slow', 'step', where 'fwd' indicates normal speed) and stepping rate, **stepRate**, which is an integer between 0 and 255. The laser disk player will stop when the given chapter or frame (depending on the mode of the

laser disk player), **position**, is reached. The ending position is an integer between 0 and 79 (if the mode is 'chapter') or between 0 and 54000 (if the mode is 'frame'). The default speed is normal and the default stepping rate is 0.

**psceneable on/off**

Send a command to the laser disk player to enable ('on') or disable ('off') PSC mode.

**repeat position [speed [repeatCount [stepRate]]]**

Send a command to the laser disk player to begin playing the laser disk, at the given **speed** ('fwd', 'fast', 'slow', 'step', where 'fwd' indicates normal speed) and stepping rate, **stepRate**, which is an integer between 0 and 255. The laser disk player will stop when the given chapter or frame (depending on the mode of the laser disk player), **position**, is reached, and it will then repeat the sequence played for the number of times given by **repeatCount**. The ending position is an integer between 0 and 79 (if the mode is 'chapter') or between 0 and 54000 (if the mode is 'frame'). The number of times is between 0 and 15. The default speed is normal, the default stepping rate is 0 and the default number of times is 1.

**reset**

Send a 'reset' command to the laser disk player.

**search new-position**

Send a command to the laser disk player to move the current position to the given chapter or frame (depending on the mode of the laser disk player), **new-position**. The new position is an integer between 0 and 79 (if the mode is 'chapter') or between 0 and 54000 (if the mode is 'frame').

**show on/off**

Send a command to the laser disk player to enable ('on') or disable ('off') the addition of the display of the current position to the outgoing video signal.

**sound theChannel on/off**

Send a command to the laser disk player to enable ('on') or disable ('off') the sound signal in the given channel, **theChannel**. The channel is either '1' or '2'.

**status**

Send a command to the laser disk player requesting the key mode status and the command status.

**step&still [fwdOrRev]**

Send a command to the laser disk player to move the current position one frame forward ('fwd') or reverse ('rev'). The default direction is forward.

**still**

Send a command to the laser disk player to freeze play.

**stop**

Send a command to the laser disk player to stop playing.

**xreset**

Unconditionally send a 'reset' command to the laser disk player. This has the side effect of changing the mode to 'frame'.

**Comments:**

Figure 3 shows how to connect an *ldp1550* object to a *serial??* object. Figure 4 shows how to connect an *ldp1550* object to a *serialX* object.

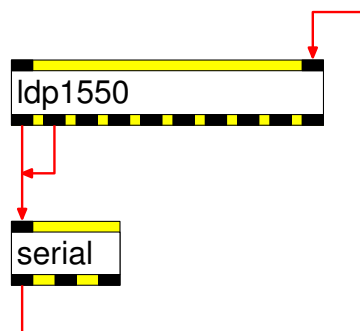


Figure 3: OS X: Connecting an *ldp1550* object to a *serial* object

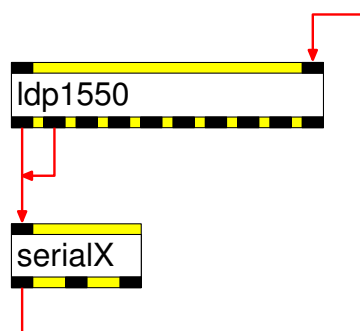
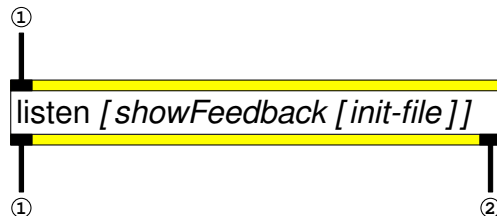


Figure 4: OS 9: Connecting an *ldp1550* object to a *serialX* object

## *listen*



**Description of object:** *listen* is an interface to the Macintosh Speech Recognition Manager.

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):**

*showFeedback* **(optional) symbol**, either 'yes', 'y', 'no' or 'n' to indicate whether to use the Apple Speech Recognition window for feedback

*init-file* **(optional) symbol**, the name of the language file to load initially

**Inlet(s):**

1 **list/bang**, the command input

**Outlet(s):**

1 **list**, the status or response. Status messages (triggered by a '**status**' command) appear as a two element list, starting with the symbol 'status'; response messages appear as a list, starting with the symbol 'result'.

2 **bang**, an error was detected

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**



**bang**

Return the previous result, if any.

**load filename**

The currently loaded language model will be set to the contents of the named language file.

**recognize phrase-list**

The currently loaded language model will be replaced by the simplified model, **phrase-list**. **phrase-list** consists of a sequence of alternate phrases, separated by the character '|'. Phrases are a sequence of one or more English words, each followed by an (optional) list of atoms that are output when this word is matched, preceeded by the character '{' and followed by the character '}'. Each word in a phrase can be followed by an optional modifier, which begins with the character '[', followed by one or more 'O', 'o', 'R' or 'r' characters and ending with the character ']'. The characters 'R' and 'r' indicate that the word that they follow can be repeated, and the characters 'O' and 'o' indicate that the word that they follow is optional.

**start**

If a language model has been loaded, listening will be enabled.

**status**

The state of the *listen* object (idle, loaded or stopped) will be reported.

**stop**

Listening will be disabled.

**File format:**

The language file describes the spoken phrases that the *listen* object will recognize and is composed of two sections:

- 1) the top-level model—a single symbol
- 2) the model descriptions—a list of statements describing the elements of the language.

Comments start with the '#' character and end with the ';' character.

The model descriptions have the following form:

- a) a linguistic symbol, which is a sequence of non-blank characters, preceeded by the character '<' and followed by the character '>'
- b) an (optional) list of atoms that are output when this linguistic symbol is matched, preceeded by the character '{' and followed by the character '}'
- c) the symbol '='
- d) one or more phrases, separated by the character '|', which indicates that the phrases are alternatives
- e) the character ';'.

Phrases are a sequence of one or more elements, where an element is:

- a) a linguistic symbol or
- b) an English word, followed by an (optional) list of atoms that are output when this word is matched, preceeded by the character '{' and followed by the character '}'

Each element in a phrase can be followed by an optional modifier, which begins with the character '[', followed by one or more 'O', 'o', 'R' or 'r' characters and ending with the character ']'. The characters 'R' and 'r' indicate that the element that they follow can be repeated, and the characters 'O' and 'o' indicate that the element that they follow is optional.

Note that the top-level model is represents the most complex sentence that is expected. Each model can only have one description, and each model must be used in at least one other model. When a sentence is recognized, the optional lists of atoms that are associated with the matching words and linguistic symbols will be output as a single list, preceeded by the symbol 'result'.

**Comments:** Note that there can only be one *listen* object in *Max* at any time. As well, *Max/MSP* by default grabs the Sound Input Manager when it starts up, so that it is necessary to make it release the Sound Input Manager before using an *listen* object.

```
#File: listen_data_1;
<CallSomeone>
<CallSomeone> { call } = Call <Person> at <Place> ;
<Person> = Matt { MJ } | Jason { JH } | Chris { CD } ;
<Place> = Work { 0 } | Home { 1 } | the office { 2 } ;
# The following spoken phrases result in the indicated output:
#call matt at home -> call MJ 1
#call jason at the office -> call JH 2
#call chris at work -> call CD 0
```

Figure 5: An example language file for a *listen* object

```

#File: listen_data.2;
# Top-level model;;
<Number>
# Models;;
<Number> = zero { 0 } | <LowNumber> | <Century> and [ o ] <LowNumber> [ o ] ;
<LowNumber> = <LowUnits> | <LowTeens> | <LowDecades> <LowUnits> [ o ] ;
<LowUnits> = one { 1 } | two { 2 } | three { 3 } | four { 4 } | five { 5 } |
    six { 6 } | seven { 7 } | eight { 8 } | nine { 9 } ;
<LowTeens> = ten { 10 } | eleven { 11 } | twelve { 12 } | thirteen { 13 } |
    fourteen { 14 } | fifteen { 15 } | sixteen { 16 } | seventeen { 17 } |
    eighteen { 18 } | nineteen { 19 } ;
<LowDecades> = twenty { 20 } | thirty { 30 } | forty { 40 } | fifty { 50 } |
    sixty { 60 } | seventy { 70 } | eighty { 80 } | ninety { 90 } ;
<Century> = <MidNumber> hundred [ o ] | <HighUnits> thousand ;
<MidNumber> = <MidUnits> | <MidTeens> | <MidDecades> <MidUnits> [ o ] ;
<MidUnits> = one { 100 } | two { 200 } | three { 300 } | four { 400 } |
    five { 500 } | six { 600 } | seven { 700 } | eight { 800 } |
    nine { 900 } ;
<MidTeens> = ten { 1000 } | eleven { 1100 } | twelve { 1200 } | thirteen { 1300 } |
    fourteen { 1400 } | fifteen { 1500 } | sixteen { 1600 } |
    seventeen { 1700 } | eighteen { 1800 } | nineteen { 1900 } ;
<MidDecades> = twenty { 2000 } | thirty { 3000 } | forty { 4000 } |
    fifty { 5000 } | sixty { 6000 } | seventy { 7000 } | eighty { 8000 } |
    ninety { 9000 } ;
<HighUnits> = one { 1000 } | two { 2000 } | three { 3000 } | four { 4000 } |
    five { 5000 } | six { 6000 } | seven { 7000 } | eight { 8000 } |
    nine { 9000 } ;
# This allows for several alternate representations of the same number;;
# 1492 = fourteen ninety two, or one thousand four hundred ninety two, or ;
#     fourteen hundred and ninety two or other variants where the word ;
#     'and' may or may not appear.;

```

Figure 6: Another example language file for a *listen* object

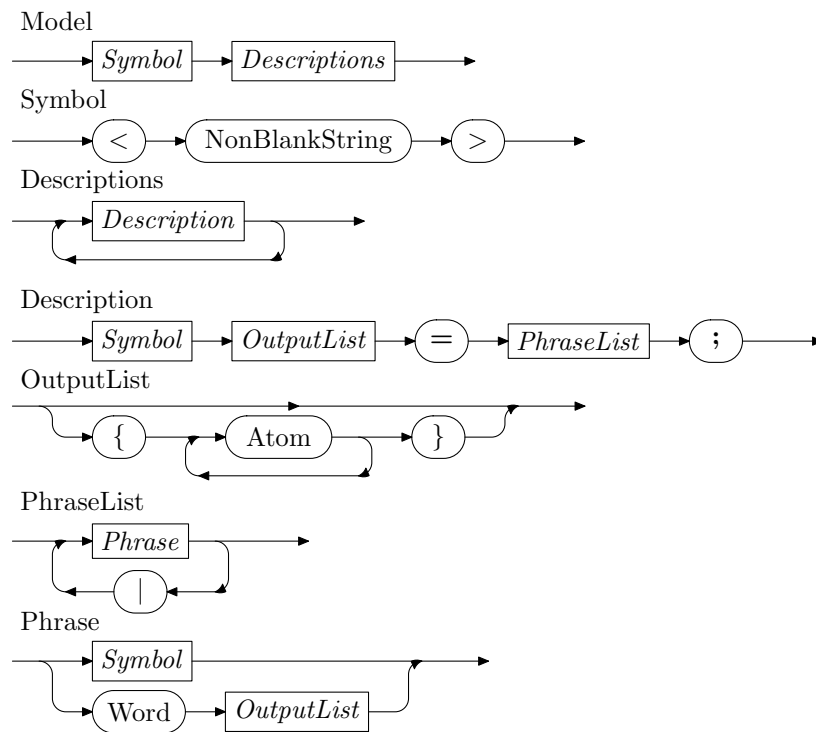
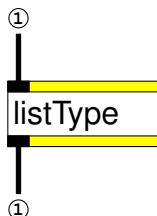


Figure 7: Syntax diagram for language files

## *listType*



**Description of object:** *listType* returns a numeric code corresponding to the value it receives.

**Object created:** October 2000

**Current version:** 1.0.3

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Lists

**Argument(s):** none

**Inlet(s):**

**1** anything, any value

**Outlet(s):**

**1 integer**, the code for the input (unknown = 0, non-list = 1, empty list = 2, integer list = 3, float list = 4, numeric list<sup>4</sup> = 5, symbol list = 6, mixed list<sup>5</sup> = 7, list with unknowns<sup>6</sup> = 8)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

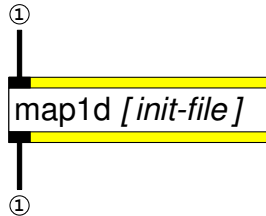
---

<sup>4</sup>A numeric list contains both integer and float values.

<sup>5</sup>A mixed list contains both numeric values and symbols.

<sup>6</sup>A list with unknowns contains one or more unrecognizable values.

## *map1d*



**Description of object:** *map1d* maps its input to one of a sequence of ranges and returns the set of values associated with the range.

**Object created:** November 2000

**Current version:** 1.0.6

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*init-file* (optional) **symbol**, the name of the map file to load initially

**Inlet(s):**

**1 integer/float**, the command or data input

**Outlet(s):**

**1 list**, the retrieved data, the previous result (if a '**bang**' is received), the number of loaded ranges, the description of an individual range or the value of an element of a range. Results or retrieved data appear as a list starting with the symbol 'result'; the number of loaded ranges appear as a two element list, starting with the symbol 'count', range descriptions appear as a list starting with the symbol 'range' and range element values appear as a list starting with the symbol 'value'.

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**add list**

Add a range to the end of the loaded set of ranges. The range is in the form: *'\*' or left-bracket lower-value upper-value right-bracket output*, where *left-bracket* is either '[' or '(' and *right-bracket* is either ']' or ')'; *lower-value* is a floating-point number, an integer or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *upper-value* is a floating-point number, an integer or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input or the symbol *'\$\$'* to indicate the offset of the input from the *lower-value* of the matching range. If the *lower-value* is unbounded, the input is returned rather than the offset.

**after index list**

Add a range after the range with the given index. The range is in the form: *'\*' or left-bracket lower-value upper-value right-bracket output*, where *left-bracket* is either '[' or '(' and *right-bracket* is either ']' or ')'; *lower-value* is a floating-point number, an integer or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *upper-value* is a floating-point number, an integer or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input or the symbol *'\$\$'* to indicate the offset of the input from the *lower-value* of the matching range. If the *lower-value* is unbounded, the input is returned rather than the offset.

**bang**

Return the previous result, if any, as a list starting with the symbol *'result'*.

**before index list**

Add a range before the range with the given index. The range is in the form: *'\*' or left-bracket lower-value upper-value right-bracket output*, where *left-bracket* is either '[' or '(' and *right-bracket* is either ']' or ')'; *lower-value* is a floating-point number, an integer or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *upper-value* is a floating-point number, an integer or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input or the symbol *'\$\$'* to indicate the offset of the input from the *lower-value* of the matching range. If the *lower-value* is unbounded, the input is returned rather than the offset.

**clear**

The currently loaded set of ranges is removed.

**count**

The number of currently loaded ranges is returned as a two element list, starting with the symbol *'count'*.

**delete index**

Removes the range with the given index from the loaded set of ranges.

**dump**

Retrieves all the ranges as a sequence of lists starting with the symbol *'range'*. The second element of each list is the index, and the remainder of the list is in the form *'\*' or left-bracket lower-value upper-value right-bracket output*, where *left-bracket* is either '[' or '(' and *right-bracket* is either ']' or ')'; *lower-value* is a floating-point number or the symbol *'-inf'*, which indicates an unbounded value; *upper-value*, is a floating-point number or the symbol *'inf'*, which indicates an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is the list of values which will be returned on a successful match.

**float**

The given value is compared to the loaded ranges. When a match is found, the output portion of the matching range is returned, prefixed with the symbol 'result'.

**get index edge**

Returns the given edge ('upper' or 'lower') of the given index as a two element list, starting with the symbol 'value'. A "don't care" value is returned as the symbol '\*'.

**integer**

The given value is compared to the loaded ranges. When a match is found, the output portion of the matching range is returned, prefixed with the symbol 'result'.

**load filename**

The currently loaded set of ranges will be set to the contents of the named map file.

**replace index list**

Replace the range with the given index. The range is in the form: '\*' or *left-bracket lower-value upper-value right-bracket output*, where *left-bracket* is either '[' or '(' and *right-bracket* is either ']' or ')'; *lower-value* is a floating-point number, an integer or one of the symbols '-inf' or '-∞', which indicate an unbounded value; *upper-value* is a floating-point number, an integer or one of the symbols 'inf', '+inf', '∞' or '+∞', which indicate an unbounded value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol '\$' to indicate the input or the symbol '\$\$' to indicate the offset of the input from the *lower-value* of the matching range. If the *lower-value* is unbounded, the input is returned rather than the offset.

**set index edge value**

Replace the given edge ('lower' or 'upper') of the range with the given index. *value* is a floating-point number, an integer, or one of the symbols 'inf', '+inf', '∞' or '+∞', '-inf' or '-∞', which indicate an unbounded value. If *edge* is 'lower', the symbol '-inf' or '-∞' can appear; if *edge* is 'upper', the symbol 'inf', '+inf', '∞' or '+∞' can appear. Note that a "don't care" value cannot be replaced using this command.

**show index**

Retrieves the range with the given index, as a list starting with the symbol 'range'. The second element of the list is the index, and the remainder of the list is in the form '\*' or *left-bracket lower-value upper-value right-bracket output*, where *left-bracket* is either '[' or '(' and *right-bracket* is either ']' or ')'; *lower-value* is a floating-point number or the symbol '-inf', which indicates an unbounded value; *upper-value*, is a floating-point number or the symbol 'inf', which indicates an unbounded value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is the list of values which will be returned on a successful match.

**verbose [on/off]**

Range check tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**File format:**

The map file is composed of a set of ranges. Comments start with the '#' character and end with the ';' character. Ranges are either open (don't include the boundary value) or closed (the boundary value is included), and have boundary values that are integers, floating-point numbers, or infinities. An open range is indicated by a parenthesis, or round bracket, and a closed range by a square bracket. A range declaration is in the following form:

'\*' or *left-bracket lower-value upper-value right-bracket output*

*left-bracket* is either '[' or '(' and *right-bracket* is either ']' or ')'; *lower-value* is a floating-point number,



an integer or one of the symbols ‘ $-\text{inf}$ ’ or ‘ $-\infty$ ’, which indicate an unbounded value; *upper-value* is a floating-point number, an integer or one of the symbols ‘ $\text{inf}$ ’, ‘ $+\text{inf}$ ’, ‘ $\infty$ ’ or ‘ $+\infty$ ’, which indicate an unbounded value. The symbol ‘ $*$ ’ indicates a “don’t care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbols ‘ $\$$ ’ or ‘ $\$x$ ’ to indicate the input or the symbol ‘ $\$\$$ ’ or ‘ $\$\$x$ ’ to indicate the offset of the input from the *lower-value* of the matching range. If the *lower-value* is unbounded, the input is returned rather than the offset. Note that the order of the range declarations is critical—the first range that matches the input is used, and overlaps between range declarations are ignored.

```
#File: map_file_1d;
# Each line is terminated with a semicolon;
# A comment starts with a '#' character;
# Note that white space is critical around symbols and operators;
# Mappings;
# The format is: <open> <lower> <upper> <close> <output> ;
# where <open> is either '[' or '(' and <close> is either ']' or ')';
# <lower> is a number or the symbol '-∞' (option-5) or '-inf';
# <upper> is a number or the symbol '∞' (option-5) or '+∞' or;
# 'inf' or '+inf';
# The bracketed pair can be replaced by the symbol * to indicate;
# a don't care value;
# <output> is what to return on a match;
# <output> can contain the symbol '$' which is replaced by the input;
# value or the symbol '$$' which is replaced by the offset of the input;
# value from <lower> - the input value is returned instead of the;
# offset if <lower> is unbounded;
[ -20 20 ] alpha $ ;
( 20 30 ] beta ;
( -∞ -20 ) gamma ;
( 30 inf ) delta $$ ;
```

Figure 8: An example map file for a *map1d* object

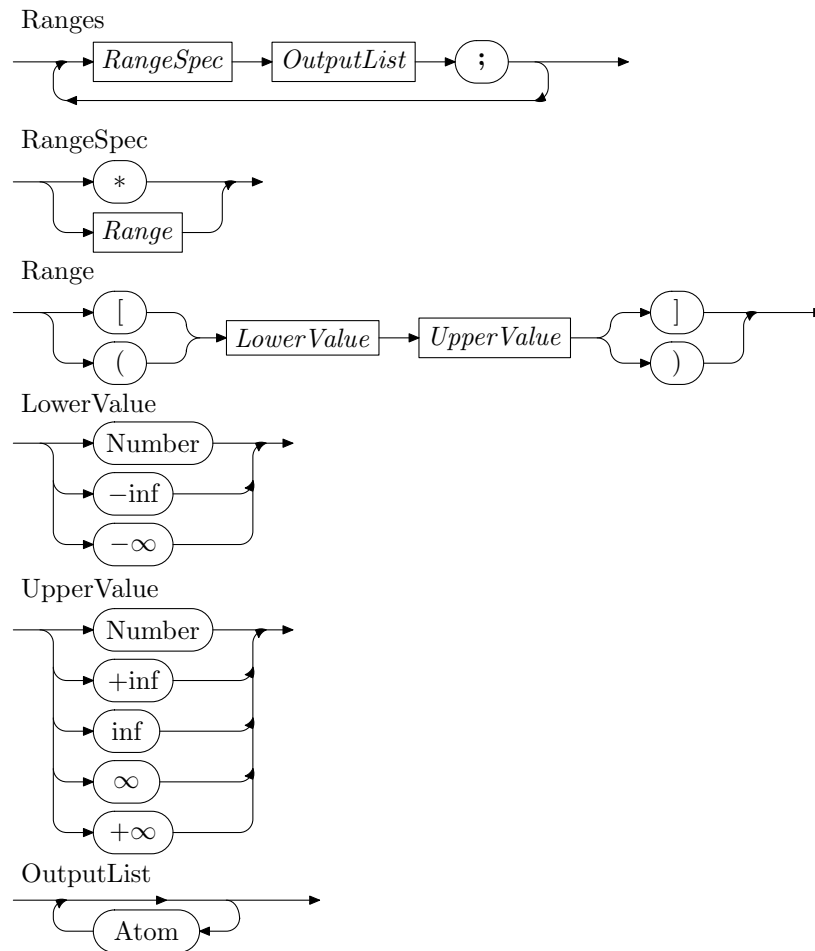
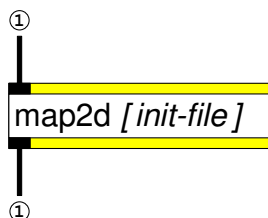


Figure 9: Syntax diagram for 1-D map files

## *map2d*



**Description of object:** *map2d* maps its input to one of a sequence of ranges and returns the set of values associated with the range.

**Object created:** November 2000

**Current version:** 1.0.6

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*init-file* (optional) **symbol**, the name of the map file to load initially

**Inlet(s):**

**1 list**, the command or data input

**Outlet(s):**

**1 list**, the retrieved data, the previous result (if a '**bang**' is received), the number of loaded ranges, the description of an individual range or the value of an element of a range. Results or retrieved data appear as a list starting with the symbol 'result'; the number of loaded ranges appear as a two element list, starting with the symbol 'count', range descriptions appear as a list starting with the symbol 'range' and range element values appear as a list starting with the symbol 'value'.

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**add list**

Add a range to the end of the loaded set of ranges. The range is in the form: *'\*' or bracket1 left right bracket2 '\*' or bracket3 bottom top bracket4 output*, where *bracket1* and *bracket3* are either '[' or '(' and *bracket2* and *bracket4* are either ']' or ')'; *left* and *bottom* are floating-point numbers, integers or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *right* and *top* are floating-point numbers, integers or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don’t care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input values or the symbol *'\$\$'* to indicate the offset of the input values from the vector (*left bottom*) of the matching range. If *left* or *bottom* is unbounded, the corresponding input value is returned rather than the offset.

**after index list**

Add a range after the range with the given index. The range is in the form: *'\*' or bracket1 left right bracket2 '\*' or bracket3 bottom top bracket4 output*, where *bracket1* and *bracket3* are either '[' or '(' and *bracket2* and *bracket4* are either ']' or ')'; *left* and *bottom* are floating-point numbers, integers or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *right* and *top* are floating-point numbers, integers or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don’t care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input values or the symbol *'\$\$'* to indicate the offset of the input values from the vector (*left bottom*) of the matching range. If *left* or *bottom* is unbounded, the corresponding input value is returned rather than the offset.

**bang**

Return the previous result, if any, as a list starting with the symbol *'result'*.

**before index list**

Add a range before the range with the given index. The range is in the form: *'\*' or bracket1 left right bracket2 '\*' or bracket3 bottom top bracket4 output*, where *bracket1* and *bracket3* are either '[' or '(' and *bracket2* and *bracket4* are either ']' or ')'; *left* and *bottom* are floating-point numbers, integers or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *right* and *top* are floating-point numbers, integers or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don’t care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input values or the symbol *'\$\$'* to indicate the offset of the input values from the vector (*left bottom*) of the matching range. If *left* or *bottom* is unbounded, the corresponding input value is returned rather than the offset.

**clear**

The currently loaded set of ranges is removed.

**count**

The number of currently loaded ranges is returned as a two element list, starting with the symbol *'count'*.

**delete index**

Removes the range with the given index from the loaded set of ranges.

**dump**

Retrieves all the ranges as a sequence of lists starting with the symbol *'range'*. The second element of each list is the index, and the remainder of the list is in the form *'\*' or bracket1 left right bracket2 '\*' or bracket3 bottom top bracket4 output*, where *bracket1* and *bracket3* are either '[' or '(' and *bracket2* and *bracket4* are either ']' or ')'; *left* and *bottom* are floating-point numbers or the symbol *'-inf'*, which indicates an unbounded value; *right* and *top* are floating-point numbers or the symbol *'inf'*, which indicates an unbounded

value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is the list of values which will be returned on a successful match.

### ***float/integer float/integer***

The given pair of values (either floating-point or integer) is compared to the loaded ranges. When a match is found, the output portion of the matching range is returned, prefixed with the symbol 'result'.

### **get index edge**

Returns the given edge ('left', 'right', 'top' or 'bottom') of the given index as a two element list, starting with the symbol 'value'. A "don't care" value is returned as the symbol '\*'.

### **load filename**

The currently loaded set of ranges will be set to the contents of the named map file.

### **replace index list**

Replace the range with the given index. The range is in the form: '\*' or *bracket1 left right bracket2 '\*'* or *bracket3 bottom top bracket4 output*, where *bracket1* and *bracket3* are either '[' or '(' and *bracket2* and *bracket4* are either ']' or ')'; *left* and *bottom* are floating-point numbers, integers or one of the symbols '-inf' or '-∞', which indicate an unbounded value; *right* and *top* are floating-point numbers, integers or one of the symbols 'inf', '+inf', '∞' or '+∞', which indicate an unbounded value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol '\$' to indicate the input values or the symbol '\$\$' to indicate the offset of the input values from the vector (*left bottom*) of the matching range. If *left* or *bottom* is unbounded, the corresponding input value is returned rather than the offset.

### **set index edge value**

Replace the given edge ('left', 'right', 'top' or 'bottom') of the range with the given index. *value* is a floating-point number, an integer, or one of the symbols 'inf', '+inf', '∞' or '+∞', '-inf' or '-∞', which indicate an unbounded value. If *edge* is 'left' or 'bottom', the symbol '-inf' or '-∞' can appear; if *edge* is 'right' or 'top', the symbol 'inf', '+inf', '∞' or '+∞' can appear. Note that a "don't care" value cannot be replaced using this command.

### **show index**

Retrieves the range with the given index, as a list starting with the symbol 'range'. The second element of the list is the index, and the remainder of the list is in the form '\*' or *bracket1 left right bracket2 '\*'* or *bracket3 bottom top bracket4 output*, where *bracket1* and *bracket3* are either '[' or '(' and *bracket2* and *bracket4* are either ']' or ')'; *left* and *bottom* are floating-point numbers or the symbol '-inf', which indicates an unbounded value; *right* and *top* are floating-point numbers or the symbol 'inf', which indicates an unbounded value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is the list of values which will be returned on a successful match.

### **verbose [on/off]**

Range check tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

## **File format:**

The map file is composed of a set of ranges. Comments start with the '#' character and end with the ';' character. Ranges are either open (don't include the boundary value) or closed (the boundary value is included), and have boundary values that are integers, floating-point numbers, or infinities. An open range is indicated by a parenthesis, or round bracket, and a closed range by a square bracket. A range declaration is in the following form:

*‘\*’ or bracket1 left right bracket2 ‘\*’ or bracket3 bottom top bracket4 output*

*bracket1* and *bracket3* are either '[' or '(' and *bracket2* and *bracket4* are either ']' or ')'; *left* and *bottom* are floating-point numbers, integers or one of the symbols  $-\text{inf}$  or  $-\infty$ , which indicate an unbounded value; *right* and *top* are floating-point numbers, integers or one of the symbols  $\text{inf}$ ,  $+\text{inf}$ ,  $\infty$  or  $+\infty$ , which indicate an unbounded value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol '\$' to indicate the input values or the symbol '\$\$' to indicate the offset of the input values from the vector (*left bottom*) of the matching range. The output list can also contain the symbols '\$x' or '\$y' to indicate the first or second input value, or the symbols '\$\$x' or '\$\$y' to indicate the offset of the first input value from the *left* value of the matching range or the offset of the second input value from the *bottom* value of the matching range. If *left* or *bottom* is unbounded, the corresponding input value is returned rather than the offset. Note that the order of the range declarations is critical—the first range that matches the input is used, and overlaps between range declarations are ignored.

```
#File: map_file_2d;
# Each line is terminated with a semicolon;
# A comment starts with a '#' character;
# Note that white space is critical around symbols and operators;
# Mappings;
# Format: <op> <left> <right> <cl> <op> <bottom> <top> <cl> <output> ;
# where <op> is either '[' or '(' and <cl> is either ']' or ')';
# <left>, <bottom> are numbers or the symbol '-inf' (option-5) or '-∞';
# <right>, <top> are numbers or the symbol 'inf' (option-5) or '+∞' or;
# 'inf' or '+inf';
# Any of the bracketed pairs can be replaced by the symbol * to indicate;
# a don't care value;
# <output> is what to return on a match;
# <output> can contain the symbol '$' which is replaced by the input vector;
# or the symbol '$$' which is replaced by the offset of the input vector from;
# the vector (<left> <bottom>) - the input value is returned instead of the;
# offset for any elements of the vector that are unbounded;
[ -20 20 ] [ -20 20 ] alpha $ ;
( 20 30 ) ( 20 30 ) beta ;
[ 20 30 ] [ 20 30 ] beta-shadow $$ ;
( -40 40 ) ( -∞ 0 ) gamma ;
```

Figure 10: An example map file for a *map2d* object

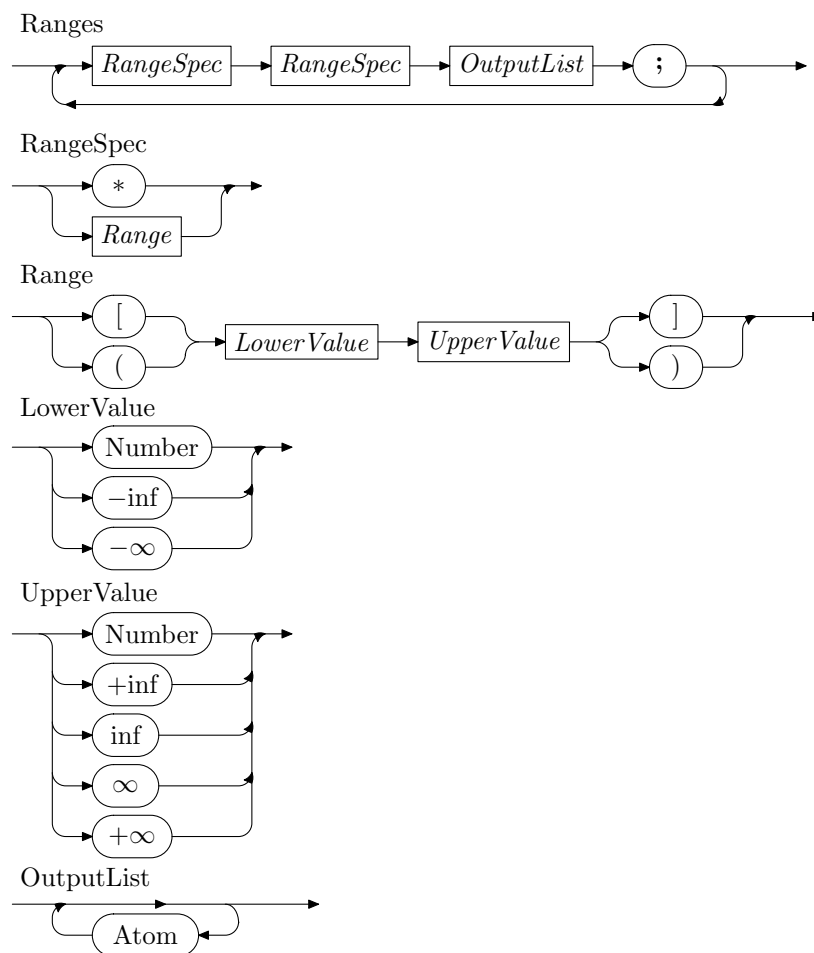
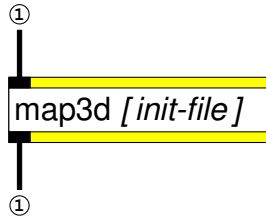


Figure 11: Syntax diagram for 2-D map files

## *map3d*



**Description of object:** *map3d* maps its input to one of a sequence of ranges and returns the set of values associated with the range.

**Object created:** April 2001

**Current version:** 1.0.6

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*init-file* (optional) **symbol**, the name of the map file to load initially

**Inlet(s):**

**1 list**, the command or data input

**Outlet(s):**

**1 list**, the retrieved data, the previous result (if a '**bang**' is received), the number of loaded ranges, the description of an individual range or the value of an element of a range. Results or retrieved data appear as a list starting with the symbol 'result'; the number of loaded ranges appear as a two element list, starting with the symbol 'count', range descriptions appear as a list starting with the symbol 'range' and range element values appear as a list starting with the symbol 'value'.

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**



**add list**

Add a range to the end of the loaded set of ranges. The range is in the form: *'\*' or b1 left right b2 '\*' or b3 bottom top b4 '\*' or b5 forward back b6 output*, where *b1*, *b3* and *b5* are either '[' or '(' and *b2*, *b4* and *b6* are either ']' or ')'; *left*, *bottom* and *forward* are floating-point numbers, integers or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *right*, *top* and *back* are floating-point numbers, integers or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input values or the symbol *'\$\$'* to indicate the offset of the input values from the vector (*left bottom forward*) of the matching range.

**after index list**

Add a range after the range with the given index. The range is in the form: *'\*' or b1 left right b2 '\*' or b3 bottom top b4 '\*' or b5 forward back b6 output*, where *b1*, *b3* and *b5* are either '[' or '(' and *b2*, *b4* and *b6* are either ']' or ')'; *left*, *bottom* and *forward* are floating-point numbers, integers or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *right*, *top* and *back* are floating-point numbers, integers or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input values or the symbol *'\$\$'* to indicate the offset of the input values from the vector (*left bottom forward*) of the matching range.

**bang**

Return the previous result, if any, as a list starting with the symbol *'result'*.

**before index list**

Add a range before the range with the given index. The range is in the form: *'\*' or b1 left right b2 '\*' or b3 bottom top b4 '\*' or b5 forward back b6 output*, where *b1*, *b3* and *b5* are either '[' or '(' and *b2*, *b4* and *b6* are either ']' or ')'; *left*, *bottom* and *forward* are floating-point numbers, integers or one of the symbols *'-inf'* or *'-∞'*, which indicate an unbounded value; *right*, *top* and *back* are floating-point numbers, integers or one of the symbols *'inf'*, *'+inf'*, *'∞'* or *'+∞'*, which indicate an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol *'\$'* to indicate the input values or the symbol *'\$\$'* to indicate the offset of the input values from the vector (*left bottom forward*) of the matching range.

**clear**

The currently loaded set of ranges is removed.

**count**

The number of currently loaded ranges is returned as a two element list, starting with the symbol *'count'*.

**delete index**

Removes the range with the given index from the loaded set of ranges.

**dump**

Retrieves all the ranges as a sequence of lists starting with the symbol *'range'*. The second element of each list is the index, and the remainder of the list is in the form *'\*' or b1 left right b2 '\*' or b3 bottom top b4 '\*' or b5 forward back b6 output*, where *b1*, *b3* and *b5* are either '[' or '(' and *b2*, *b4* and *b6* are either ']' or ')'; *left*, *bottom* and *forward* are floating-point numbers or the symbol *'-inf'*, which indicates an unbounded value; *right*, *top* and *back* are floating-point numbers or the symbol *'inf'*, which indicates an unbounded value. The symbol *'\*'* indicates a “don't care” value, which will match anything. *output* is the list of values which will be returned on a successful match.

*float/integer float/integer float/integer*

The given triple of values (either floating-point or integer) is compared to the loaded ranges. When a match is found, the output portion of the matching range is returned, prefixed with the symbol 'result'.

**get index edge**

Returns the given edge ('left', 'right', 'top', 'bottom', 'forward' or 'back') of the given index as a two element list, starting with the symbol 'value'. A "don't care" value is returned as the symbol '\*'.

**load filename**

The currently loaded set of ranges will be set to the contents of the named map file.

**replace index list**

Replace the range with the given index. The range is in the form: '\*' or *b1 left right b2 '\*' or b3 bottom top b4 '\*' or b5 forward back b6 output*, where *b1*, *b3* and *b5* are either '[' or '(' and *b2*, *b4* and *b6* are either ']' or ')'; *left*, *bottom* and *forward* are floating-point numbers, integers or one of the symbols '-inf' or '-∞', which indicate an unbounded value; *right*, *top* and *back* are floating-point numbers, integers or one of the symbols 'inf', '+inf', '∞' or '+∞', which indicate an unbounded value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol '\$' to indicate the input values or the symbol '\$\$' to indicate the offset of the input values from the vector (*left bottom forward*) of the matching range.

**set index edge value**

Replace the given edge ('left', 'right', 'top', 'bottom', 'forward' or 'back') of the range with the given index. *value* is a floating-point number, an integer, or one of the symbols 'inf', '+inf', '∞' or '+∞', '-inf' or '-∞', which indicate an unbounded value. If *edge* is 'left', 'bottom', or 'forward', the symbol '-inf' or '-∞' can appear; if *edge* is 'right', 'top' or 'back', the symbol 'inf', '+inf', '∞' or '+∞' can appear. Note that a "don't care" value cannot be replaced using this command.

**show index**

Retrieves the range with the given index, as a list starting with the symbol 'range'. The second element of the list is the index, and the remainder of the list is in the form '\*' or *b1 left right b2 '\*' or b3 bottom top b4 '\*' or b5 forward back b6 output*, where *b1*, *b3* and *b5* are either '[' or '(' and *b2*, *b4* and *b6* are either ']' or ')'; *left*, *bottom* and *forward* are floating-point numbers or the symbol '-inf', which indicates an unbounded value; *right*, *top* and *back* are floating-point numbers or the symbol 'inf', which indicates an unbounded value. The symbol '\*' indicates a "don't care" value, which will match anything. *output* is the list of values which will be returned on a successful match.

**verbose [on/off]**

Range check tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**File format:**

The map file is composed of a set of ranges. Comments start with the '#' character and end with the ';' character. Ranges are either open (don't include the boundary value) or closed (the boundary value is included), and have boundary values that are integers, floating-point numbers, or infinities. An open range is indicated by a parenthesis, or round bracket, and a closed range by a square bracket. A range declaration is in the following form:

'\*' or *b1 left right b2 '\*' or b3 bottom top b4 '\*' or b5 forward back b6 output*

*b1*, *b3* and *b5* are either '[' or '(' and *b2*, *b4* and *b6* are either ']' or ')'; *left*, *bottom* and *forward* are floating-point numbers, integers or one of the symbols '-inf' or '-∞', which indicate an unbounded value; *right*, *top* and *back* are floating-point numbers, integers or one of the symbols 'inf', '+inf', '∞'

or  $+\infty$ , which indicate an unbounded value. The symbol  $*$  indicates a “don’t care” value, which will match anything. *output* is a list of values to return on a successful match. The output list can contain the symbol  $\$$  to indicate the input values or the symbol  $\$ \$$  to indicate the offset of the input values from the vector (*left bottom forward*) of the matching range. The output list can also contain the symbols  $\$x$ ,  $\$y$  or  $\$z$  to indicate the first, second or third input value, or the symbols  $\$ \$x$ ,  $\$ \$y$  or  $\$ \$z$  to indicate the offset of the first input value from the *left* value of the matching range, the offset of the second input value from the *bottom* value of the matching range or the offset of the third input value from the *forward* value of the matching range. If *left*, *bottom* or *forward* is unbounded, the corresponding input value is returned rather than the offset. Note that the order of the range declarations is critical—the first range that matches the input is used, and overlaps between range declarations are ignored.

```
#File: map_file_3d;
# Each line is terminated with a semicolon;
# A comment starts with a '#' character;
# Note that white space is critical around symbols and operators;
# Mappings;
# Format: <o> <lft> <rht> <c> <o> <bot> <top> <c> <o> <for> <bck> <c> <out> ;
# where <o> is either '[' or '(' and <c> is either ']' or ')';
# <lft>, <bot>, <for> are numbers or the symbol '-inf' (option-5) or '-inf';
# <rht>, <top>, <bck> are numbers or the symbol 'inf' (option-5) or '+inf' or;
# 'inf' or '+inf';
# Any of the bracketed pairs can be replaced by the symbol * to indicate;
# a don't care value;
# <out> is what to return on a match;
# <out> can contain the symbol '$' which is replaced by the input vector or;
# the symbol '$$' which is replaced by the offset of the input vector from the vector;
# (<lft> <bot> <for>) - the input value is returned instead of the offset for any;
# elements of the vector that are unbounded;
[ -20 20 ] [ -20 20 ] ( 20 +inf ] alpha $ ;
( 20 30 ) ( 20 30 ) ( -inf 10 ] beta ;
[ 20 30 ] [ 20 30 ] [ 15 30 ] beta-shadow $$ ;
( -40 40 ) ( -inf 0 ) ( 12 100 ) gamma ;
```

Figure 12: An example map file for a *map3d* object

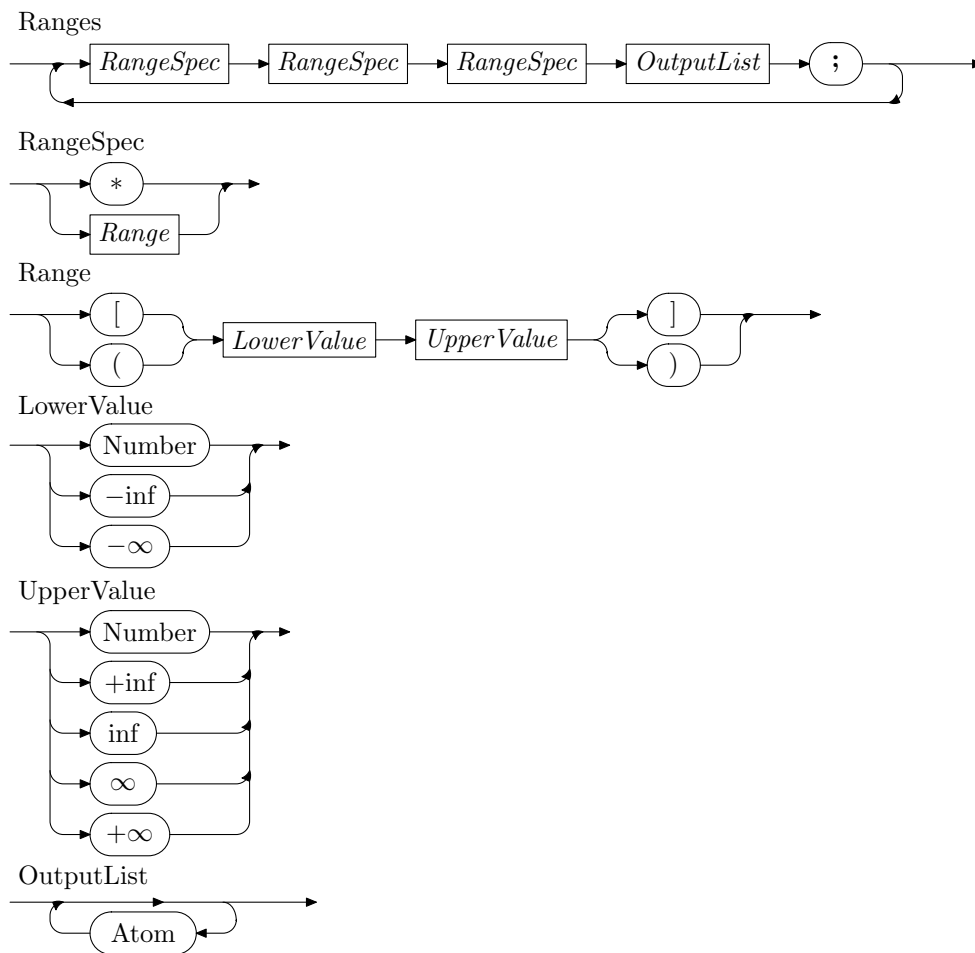
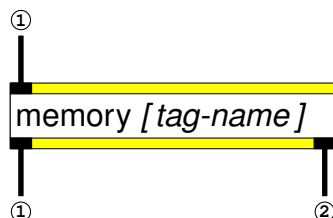


Figure 13: Syntax diagram for 3-D map files

## *memory*



**Description of object:** *memory* provides a repository for values, using an associative table to give fast access to the retained data.

**Object created:** June 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Data

**Argument(s):**

*tag-name* (optional) **symbol**, shares the associative table with all other *memory* objects having the same *tag-name*

**Inlet(s):**

1 **list**, the command input

**Outlet(s):**

1 **anything**, the retrieved data

2 **bang**, an error was detected

**Companion object(s):** none

**Standalone:** no, the object works with other *memory* objects with the same *tag-name*

**Retains state:** yes

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**clear**

Empties the associative table of all retained values.

**forget symbol**

Removes the symbol from the associative table, along with its data.

**load filename**

Clears the associative table and reads the given file into it. The file must have been created by a ‘store’ command.

**recall symbol**

Retrieves the data stored with the given symbol from the associative table.

**remember symbol [data]**

Stores the data with the given symbol in the associative table, forgetting any previous data associated with the symbol.

**store filename**

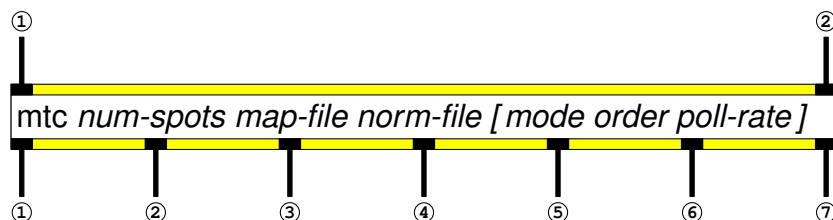
Write the associative table to the given file.

**trace [on/off]**

Associative table update tracing to the *Max* window will be enabled (‘on’), disabled (‘off’) or reversed, if no argument is given.

**Comments:**

The *memory* object was designed to address some problems that were found in attempting to use *table* objects to perform complex state sequencing.

***mtc***

**Description of object:** *mtc* is an interface to the [Tactex Controls](#) multi-touch controller (MTC). It sends commands to a *serial* or a *serialX* object, which controls the serial port that the MTC is attached to, and responds to data returned from the MTC via the *serial* or *serialX* object.

**Object created:** December 1999

**Current version:** 1.0.7

**Online help file:** no

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):**

***num-spots* integer**, the maximum number of hot spots to return

***map-file* symbol**, the map file that contains the coordinates of the sensor points for the MTC device

***norm-file* symbol**, the normalization file for the pressure values for the MTC device

***mode* (optional) symbol**, the initial processing mode (raw or cooked) that is to be used. By default, the mode is cooked.

***order* (optional) symbol**, the initial sort order (pressure, x, or y) that is to be used to prioritize the hot spots. By default, the hot spots are unordered.

***poll-rate* (optional) integer**, the rate (in milliseconds) at which the companion *serial* or *serialX* object is polled via a sample request. The default rate is 100 milliseconds between sample requests.

**Inlet(s):**

**1 list**, the command channel

**2 anything**, the output of the companion *serial* or *serialX* object

**Outlet(s):**

**1 list**, the detected hot spots in the form of floating point triples (x coordinate, y coordinate, pressure)

**2 integer**, the data has started ('0') or ended ('1'). The '0' to '1' transition preceeds the data from the MTC and the '1' to '0' transition follows the data from the MTC.

- 3 **bang**, the command has completed
- 4 **bang**, the sample request to send to the companion *serial* or *serialX* object
- 5 **anything**, the data to send to the companion *serial* or *serialX* object
- 6 **bang**, the ‘**chunk**’ request to send to the companion *serial* or *serialX* object, via a message object
- 7 **bang**, an error was detected

**Companion object(s):** works with *serialX* objects with OS 9, or *serial* objects with OS X, and can be connected to *mtcTrack* objects

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**describe**

Read the MTC descriptor string from the device (if it hasn’t already been read) and send the string to the *Max* window.

**mode symbol**

Change the processing mode (where **symbol** is ‘raw’ or ‘cooked’) that is to be used.

**order symbol**

Change the sort order (where **symbol** is ‘pressure’, ‘x’, or ‘y’) that is to be used to prioritize the hot spots.

**ping**

Perform a ‘ping’ of the MTC device to verify connectivity.

**start [normal/compressed]**

Start sampling the MTC device for data (hot spots or raw pressures). If ‘compressed’ is requested, the compressed form of data communication is used with the MTC device; if ‘normal’ is specified, or no argument is given, then the uncompressed form of data communication is used with the MTC device.

**stop**

Stop sampling the MTC device for data (hot spots or raw pressures).

**threshold new-level**

Set the pressure threshold for hot spots to **new-level**. Sensors with values above the pressure threshold are candidates for the hot spot list.

**train [start/stop]**

Start or stop the ‘training’ mode of the *mtc* object, where the high and low pressure levels for each sensor are determined to establish the dynamic range of the sensor.

**verbose [on/off]**

Tracing to the *Max* window of the messages sent will be enabled (‘on’), disabled (‘off’) or reversed, if no argument is given.

**Comments:** Figure 14 shows how to connect an *mtc* object to a *serial* object. Figure 15 shows how to connect an *mtc* object to a *serialX* object. Figure 16 is a state diagram for *mtc* objects.



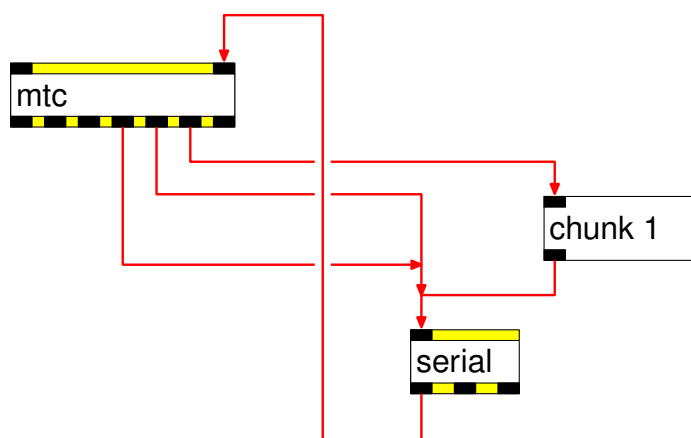


Figure 14: OS X: Connecting an *mtc* object to a *serial* object

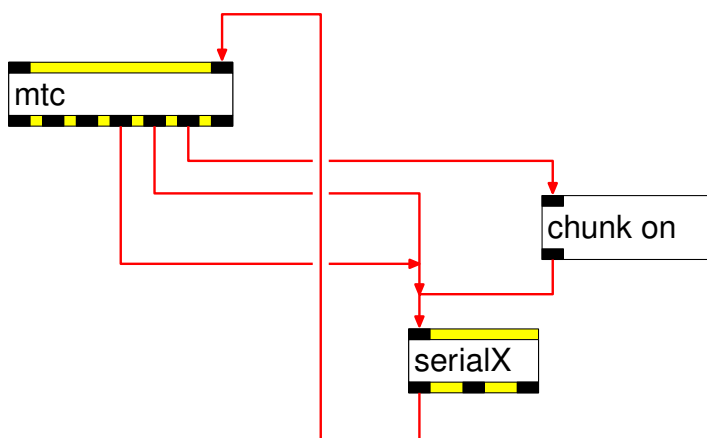


Figure 15: OS 9: Connecting an *mtc* object to a *serialX* object

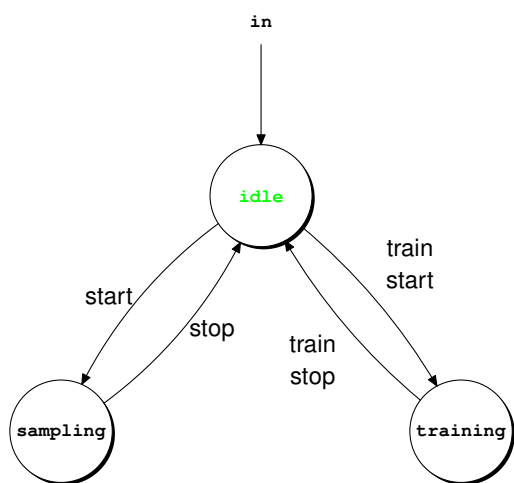
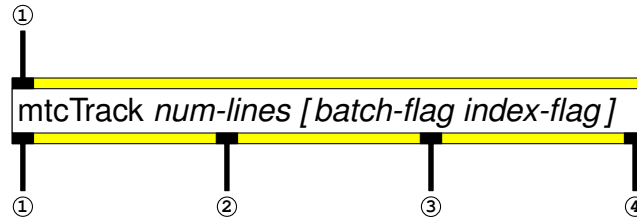


Figure 16: State diagram for *mtc* objects

## *mtcTrack*



**Description of object:** *mtcTrack* is an auxiliary object to be used with *mtc* objects. Its purpose is to convert the raw list of coordinate and pressure data into a series of lines.

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** no

**Object theme:** Miscellaneous

**Object class(es):** Lists

**Argument(s):**

***num-lines*** **integer**, the maximum number of lines to return

***batch-flag*** **(optional) symbol**, whether to output the batch number. If the word ‘batch’ appears, the batch number is prepended to each output line. By default, the batch number is not output. Batch numbers start at zero and are incremented with each set of lines returned, regardless of whether the batch number is output.

***index-flag*** **(optional) symbol**, whether to output the index of the line, relative to the other lines in the batch. If the word ‘index’ appears, the index is prepended to each output line, after the batch number, if it is present. By default, the index is not output. Indices start at zero.

**Inlet(s):**

**1 list**, the values to be converted, from the companion *mtc* object

**Outlet(s):**

**1 list**, the recognized lines, in the form of a six, seven or eight element list of numbers, consisting of the (optional) batch number, the (optional) line index (which starts at zero), the starting x coordinate, the starting y coordinate, the ending x coordinate, the ending y coordinate, the “velocity” and the “force”. The last two elements are estimates of the line’s characteristics.

**2 bang**, the last recognized line has been sent

**3 integer**, the number of lines recognized. This value is output before the lines are output, so that the lines can be collected and processed in a batch.

**4 bang**, an error was detected

<b>Companion object(s):</b>	works with <i>mtc</i> objects
<b>Standalone:</b>	yes
<b>Retains state:</b>	yes, the previously identified points
<b>Compatibility:</b>	<i>Max 3.x</i> and <i>Max 4.x</i> {OS 9 and OS X}
<b>Fat, PPC-only or 68k-only:</b>	Fat

**Command language syntax:****batch [on/off]**

Output of the batch number with each line will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**clear**

Reset the number of previously identified points, so that the next set of points received will be considered the start of a set of new lines.

**index [on/off]**

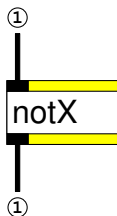
Output of the line number index with each line will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**threshold distance**

Set the maximum distance that can separate the starting and ending points of a recognized line. If **distance** is negative, there is no maximum. Use of this command reduces the thrashing that occurs when the matching points of lines are not sufficiently separated to permit unambiguous determination of the recognized lines.

<b>Comments:</b>	The <i>mtcTrack</i> object was designed to assist in working with the <i>mtc</i> object. The maximum number of lines to return should be close in value to the the maximum number of hot spots specified for the companion <i>mtc</i> object.
------------------	---

## *notX*



**Description of object:** *notX* provides the logical complement of a number or a list of numbers, returning '0' for each non-zero number and '1' for each zero number. Non-numeric values are returned without modification.

**Object created:** September 1998

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

1 **anything**, the input to be processed

**Outlet(s):**

1 **anything**, the input after complementation, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

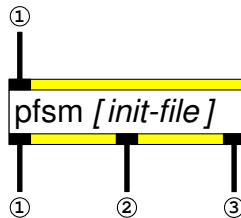
**Command language syntax:**

*bang*

Return the previous result, if any.

**Comments:** The *notX* object was created because there was a *not* object that wasn't Fat, when a Fat object was needed. It was extended to handle lists and floating point values.

## *p fsm*



**Description of object:** *p fsm* is an implementation of a Finite State Machine, with probabilistic transitions. Hence, Probabilistic Finite State Machine, or PFSM. It processes a sequence of messages against a state file, generating output as guided by the state transitions triggered by the input.

**Object created:** May 2000

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*init-file* (optional) **symbol**, the name of the state file to load initially

**Inlet(s):**

**1 integer/list**, the command input

**Outlet(s):**

**1 list**, the retrieved data

**2 bang**, a stop state was reached

**3 bang**, an error state was reached or an error was detected

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**autorestart [on/off]**

Enable ('on') or disable ('off') autorestart. If autorestart is enabled, when the *pfsm* object reaches a 'stop' state it will automatically advance to the 'start' state. In either case, a 'bang' will always be sent to the stop state outlet.

**clear**

The current state will be cleared. This is not the same as going to the 'start' state.

**describe**

The currently loaded set of transitions will be reported to the *Max* window.

**do list**

The first element in the given list is matched against the transitions available for the current state. If a match is found, the current state is set to the destination of the transition and the output portion of the transition is processed using the remainder of the given list.

**goto new-state**

The current state will be set to **new-state**, if it is in the currently loaded set of transitions.

**integer**

The given value is matched against the transitions available for the current state. If a match is found, the current state is set to the destination of the transition and the output portion of the transition is processed using an empty list as the remainder of the input.

**list anything**

The first element in the given list is matched against the transitions available for the current state. If a match is found, the current state is set to the destination of the transition and the output portion of the transition is processed using the remainder of the given list.

**load filename**

The currently loaded set of transitions will be set to the contents of the named state file.

**start**

The current state will be set to the start state for the currently loaded set of transitions and the *pfsm* object will be set to 'running'.

**status**

The state of the *pfsm* object (running or stopped, the current state, the start state and whether autorestart is enabled) will be reported to the *Max* window.

**stop**

The *pfsm* object will be stopped and the current state will be cleared.

**trace [on/off]**

State transition tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**File format:**

The state file describes the legal transitions that the *pfsm* object will perform and is composed of five sections:

- 1) the state symbols—a list of symbols enclosed in square brackets ('[' and ']')
- 2) the starting state—a single symbol
- 3) the stop states—a list of symbols enclosed in square brackets

- 4) the error states—a list of symbols enclosed in square brackets
- 5) the transitions—a list of statements describing the transitions.

The order of the transitions is critical—for each state, the first transition matching the input provided (and meeting the probability criteria, if it is a probabilistic transition) will be processed and the remaining transitions will be ignored. For this reason, transitions with specific matching values should precede transitions with wild card matching criteria.

Comments start with the '#' character and end with the ';' character.

The transitions are in one of two forms, and are terminated with the ';' character:

- 5a) current-state input -> new-state output
- 5b) current-state input -? probability new-state output

The output list can contain the symbol '\$' to indicate the matching input or the symbol '\$\$' to indicate the overflow values (any values in the input list, after the matching element) or the symbol '\$\*' to indicate the new name of the new state. The value 'input' can be any of the following special symbols:

- a) @s represents any valid symbol
- b) @n represents any valid number
- c) @r, followed by two numbers, represents a range of values
- d) '?', where ? is any printable character, represents the numeric equivalent to the character, and can appear wherever a number could appear
- e) anything else represents the value that will be matched literally

The probability is in the form of a fraction between 0.0 and 1.0

**Comments:**

The *pfsm* object was designed to address some problems that were found in attempting to use *table* objects to perform complex state sequencing. I realized that a better approach was to represent the actions as the output of an FSM, and added the probabilistic transition mechanism as a useful extension. Figure 18 is a state transition diagram for the example state file, Figure 17.



```

#File:  state-file;
# Each line is terminated with a semicolon;
# A comment starts with a '#' character;
# Note that white space is critical around symbols and operators;

# State symbols;
[ alpha omega beta test-state theta gamma ]

# Starting state;
alpha

# Stop states;
[ theta ]

# Error states;
[ omega ]

# Transitions;
# The format is:  current-state input -> new-state output ;
# or current-state input -?  probability new-state output ;
# where output can contain the symbol $ to indicate the matching input;
# the symbol $$ to indicate the overflow values or the symbol;
# $* to indicate the new state;
# The special symbol @s represents any valid symbol on input;
# The special symbol @n represents any valid number on input;
# The special symbol @r represents a range of values (the next two numbers;
# in the input);
# The special form '?' represents the numeric equivalent to the character ?;
# for input;
# The probability is in the form of a fraction between 0 and 1;

alpha 42 -> beta $ ;
alpha 'a -> beta $;
alpha @r 'b 'g -> theta $;
test garbage -> test ;
beta blorg -> alpha ;
beta blirg -> theta $ ;
alpha @s -> gamma $$ $;
gamma @n -> beta chuck you $ times;
theta @s -> theta symbol;
theta @n -> theta number;
gamma @s -?  0.30 beta whoo hoo; #probabilistic transition to beta;
gamma @s -> gamma; #handle non-branch to beta;

```

Figure 17: An example state file for a *pfsm* object

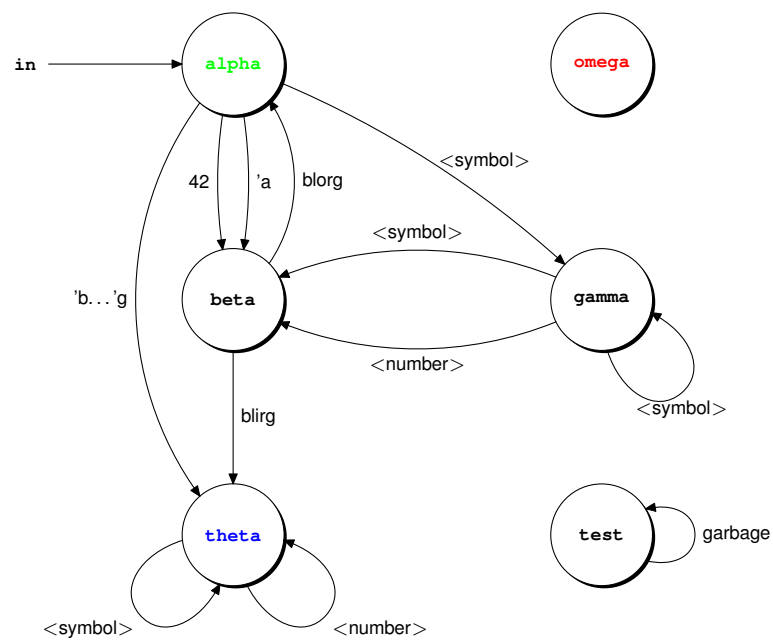


Figure 18: State transition diagram for the example state file

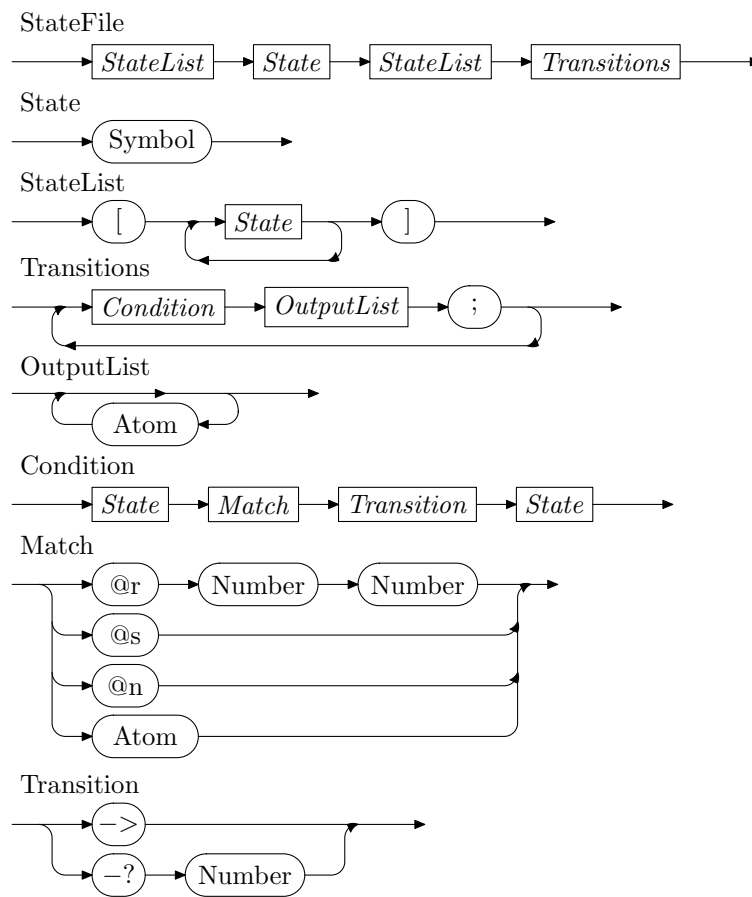
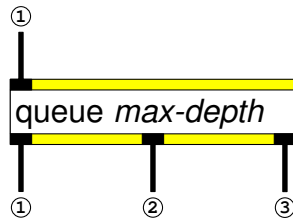


Figure 19: Syntax diagram for state files

## *queue*



**Description of object:** *queue* is an implementation of first-in-first-out queues. Unlike conventional queues, the *queue* object can store lists as well as single values on each level.

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Data

### **Argument(s):**

**tag-name integer**, the maximum number of elements that can be held in the queue. This is also known as its depth. A depth of zero is interpreted as a queue that can hold an unlimited number of elements.

### **Inlet(s):**

**1 list**, the command input

### **Outlet(s):**

**1 anything**, the retrieved data

**2 integer**, the depth of the queue (the number of elements held in the queue)

**3 bang**, an error was detected

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

### **Command language syntax:**

**add anything**

Place the given data at the end of the queue. If the queue was already full, output the first element in the queue (the oldest) before adding the new data.

***bang***

Output all data held in the queue in the order in which it was received.

**clear**

Remove all data from the queue.

**depth**

Return the depth of the queue.

**fetch**

Return the first element of the queue, without removing it.

**pull**

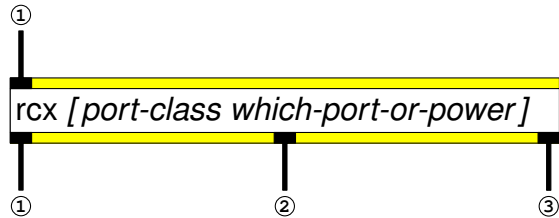
Return the first element of the queue and remove it from the queue.

**setdepth new-depth**

Set the maximum depth of the queue to new-depth. If the new maximum depth is zero, the contents of the queue are left alone. If the new maximum depth is less than the current depth, all the data is removed from the queue.

**trace [on/off]**

Queue update tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

***rcx***

**Description of object:** *rcx* is an interface to the [LEGO](#) MINDSTORMS RCX device. For Mac OS X, it uses direct operations with a LEGO USB Tower to connect to the device; for Mac OS 9, it uses the Ghost API from LEGO to connect to the device, using either a USB or serial port. Note that it does not use the *serialX* or *serial* objects for this communication.

**Object created:** April 2002

**Current version:** 1.0.5

**Online help file:** no

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):**

***port-class* (optional) symbol**, the kind of Ghost port ('usb' or 'serial') that is to be used. By default, the port-class is USB. For Mac OS X, only USB is supported.

***which-port* (optional) symbol**, the name of the port within the port-class that is to be used, or the power-level for USB (for Mac OS X). By default, the first available port is used (for Mac OS 9) and low power is used (for Mac OS X).

**Inlet(s):**

**1 list**, the command channel

**Outlet(s):**

**1 list**, the data requested from the RCX device. The individual commands indicate the format of their results.

**2 bang**, the command has completed

**3 bang**, an error was detected

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x*

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

**clearmemory**

Clears all programs out of the RCX.

**clearsensor index**

Clear the value of the given sensor, where **index** is between 1 and 3.

**clearsound**

Clear the sound buffer. See also the **'mute'** command.

**continuetask index**

Resume the execution of the given task, where **index** is between 1 and 10.

**floatoutput set**

Set the given set of outputs to "coast", where **set** is between 1 and 7 (the individual outputs are encoded as 1, 2 and 4, which are then added together to get the set of outputs to change).

**getallsensors**

Read the values, types and modes of all three sensors of the RCX and return them as a ten element list, starting with the symbol 'allsensors', followed by triples (value as an integer, type as one of 'nosensor', 'switch', 'temperature', 'reflection', 'angle' or 'unknown' and mode as one of 'raw', 'boolean', 'transition', 'periodcount', 'percent', 'celsius', 'fahrenheit', 'anglestep' or 'unknown') for each of the sensors, in order of the sensor number.

**getallvariables**

Read the values of all thirty-two variables of the RCX and return them as a list, starting with the symbol 'allvariables' and followed by the value of each of the variables.

**getbattery**

Read the battery level from the RCX and return it as a two element list, starting with the symbol 'battery', followed by a floating-point value representing the voltage.

**getsensormode index**

Read the mode of the given sensor, where **index** is between 1 and 3, and return it as a two element list, starting with the symbol 'sensormode', followed by the current value of the sensor as one of 'raw', 'boolean', 'transition', 'periodcount', 'percent', 'celsius', 'fahrenheit', 'anglestep' or 'unknown'.

**getsensortype index**

Read the type of the given sensor, where **index** is between 1 and 3, and return it as a two element list, starting with the symbol 'sensortype', followed by the current type of the sensor as one of 'nosensor', 'switch', 'temperature', 'reflection', 'angle' or 'unknown'.

**getsensorvalue index**

Read the value of the given sensor, where **index** is between 1 and 3, and return it as a two element list, starting with the symbol 'sensor', followed by the current value of the sensor as an integer.

**getslot**

Read the index of the active program and return it as a two element list, starting with the symbol 'slot', followed by the index as an integer between 1 and 5.

**getvariable index**

Read the value of the given variable, where **index** is between 1 and 32, and return it as a two element list, starting with the symbol 'variable', followed by the current value of the variable as an integer.

**getversion**

Read the version number of the ROM and the firmware from the RCX and return it as a three element list, starting with the symbol 'version', followed by the ROM version number and the firmware version number, represented as integers.

**mute**

Clear the sound buffer of the RCX and ignore future sound requests. See also the '**clearsound**' command.

**playsound sound**

Play the given system sound, which is either 'keyclick', 'beep', 'sweeplown', 'sweepup', 'error' or 'fast-sweep'.

**playtone frequency duration**

Play the given tone, where **frequency** is in Hertz and **duration** is in 100ths of a second.

**setoutputdirection set direction**

Set the direction of the given set of outputs, where **set** is between 1 and 7 (the individual outputs are encoded as 1, 2 and 4, which are then added together to get the set of outputs to change) and **direction** is either 'backward', 'reverse' or 'forward'.

**setmessage message**

Set the message buffer to the given value, where **message** is between 1 and 255.

**setoutputpower set level**

Set the power level for the given set of outputs, where **set** is between 1 and 7 (the individual outputs are encoded as 1, 2 and 4, which are then added together to get the set of outputs to change) and **level** is between 1 and 8.

**setsensormode index mode**

Set the mode of the given sensor, where **index** is between 1 and 3 and **mode** is one of 'raw', 'boolean', 'transition', 'periodcount', 'percent', 'celsius', 'fahrenheit' or 'anglestep'.

**setsensortype index type**

Set the type of the given sensor, where **index** is between 1 and 3 and **type** is one of 'nosensor', 'switch', 'temperature', 'reflection' or 'angle'.

**setslot index**

Select the given program slot as the active program, where **index** is between 1 and 5.

**setwatch hour minute**

Set the displayed time on the RCX to the given values.

**run**

Start the active program's execution.

**setvariable index value**

Set the value of the given variable, where **index** is between 1 and 32.

**startoutput set**

Set the given set of outputs on, where **set** is between 1 and 7 (the individual outputs are encoded as 1, 2 and 4, which are then added together to get the set of outputs to change).

**starttask index**

Start the execution of the given task, where **index** is between 1 and 10.

**stopalltasks**



Stop the execution of all tasks for the active program on the RCX device.

**stopoutput set**

Set the given set of outputs off, where **set** is between 1 and 7 (the individual outputs are encoded as 1, 2 and 4, which are then added together to get the set of outputs to change).

**stoptask index**

Stop the execution of the given task, where **index** is between 1 and 10.

**turnoff**

Turn off the RCX device.

**unmute**

Re-enable the processing of sound requests.

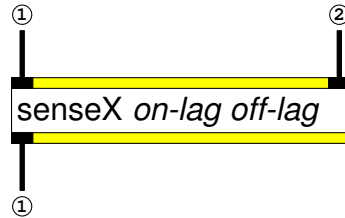
**verbose [on/off]**

Tracing to the *Max* window of the messages sent will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**Comments:**

The [MacNQC](#) development environment is recommended for developing code to be downloaded into the RCX device.

## *senseX*



**Description of object:** *senseX* outputs a pulse when two messages appear within a specified interval.

**Object created:** January 2006

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Arith/Logic/Bitwise

**Argument(s):**

***on-lag* integer**, the maximum time (in milliseconds) between two messages arriving at the right inlet that will result in an output

***off-lag* symbol**, the duration of the output pulse, in milliseconds, as well as the minimum time after recognizing a message pair before another pair can be detected

**Inlet(s):**

**1 list**, the command channel

**2 anything**, the messages being watched

**Outlet(s):**

**1 integer**, a pulse with the value '1' that is "off-lag" milliseconds in duration, when a message pair is detected; otherwise a value of '0'

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

**off integer**

Set the “off-lag” for the object to the given number of milliseconds.

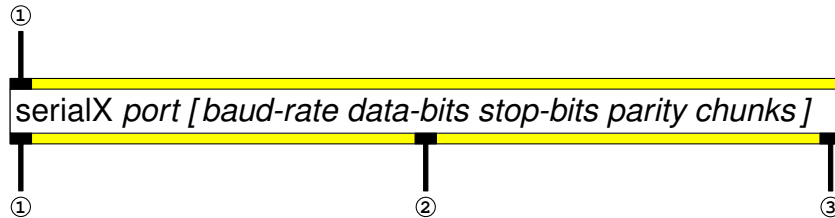
**on integer**

Set the “on-lag” for the object to the given number of milliseconds.

**Comments:**

The *senseX* object was created because there was an unsupported *sense* object that wasn't available for the current versions of *Max*, but was needed for a particular project.

## *serialX*



**Description of object:** *serialX* is an interface to the serial ports on a Macintosh. It provides functionality over and above that of the standard *serial* object.

**Object created:** June 1998

**Current version:** 1.1.4

**Online help file:** yes

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):**

**port symbol**, either 'modem', 'printer', or a single letter from 'a' to 'z' which represents the port to be connected to

**baud-rate (optional) integer/symbol**, the baud rate to set the port to. The acceptable values are: 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 14400, 19200, 28800, 38400, 57600, 115200 and 230400. Some of the baud rates can also be expressed in terms of 'kilobaud': 0.3k (or .3k), 0.6k (or .6k), 1.2k, 1.8k, 2.4k, 3.6k, 4.8k, 7.2k, 9.6k, 14.4k, 19.2k, 28.8k, 38.4k, 57.6k, 115.2k or 230.4k. The following special externally clocked rates may be available, if the serial port supports them: MIDI\_1, MIDI\_16, MIDI\_32 and MIDI\_64. These correspond to rate multipliers of 1, 16, 32 and 64, respectively. Use of these special rates forces the data bits to 8 and the stop bits to 1. Note that the character 'k' is lower case. The default baud rate is 4800.

**data-bits (optional) integer**, the number of data bits to set the port to. The value is between 5 and 9, inclusive. The default number of data bits is 8.

**stop-bits (optional) float**, the number of stop bits to set the port to. The acceptable values are: 1, 1.5 or 2. The default number of stop bits is 1.

**parity (optional) symbol**, the parity mode to set the port to. The acceptable values are: 'off'/'no'/'none', 'even' or 'odd'. The default parity mode is 'off'.

**chunks (optional) integer**, the number of characters in each chunk that will be sent when 'chunk' mode is active. The default chunk size is 10 and the maximum is 80.

**Inlet(s):**

**1 integer/list/bang**, the command input

**Outlet(s):**

- 1 integer/list**, returned characters (as single characters, if ‘chunk’ mode is inactive, or as lists of characters, if ‘chunk’ mode is active) from the port of the serial device
- 2 bang**, a break signal was seen on the port of the serial device
- 3 bang**, the break operation has completed

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the serial port settings

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 only}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:*****bang***

Return any characters received from the port of the serial device (since the last time a ‘**bang**’ command was received), in the form of one or more lists of characters, if ‘chunk’ mode is active, or as single characters, if ‘chunk’ mode is inactive. Any detected ‘break’ signals from the port of the serial device will also be reported.

**baud integer/symbol**

Change the baud rate of the port to the given value. The acceptable values are: 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 14400, 19200, 28800, 38400, 57600, 115200 and 230400. Some of the baud rates can also be expressed in terms of ‘kilobaud’: 0.3k (or .3k), 0.6k (or .6k), 1.2k, 1.8k, 2.4k, 3.6k, 4.8k, 7.2k, 9.6k, 14.4k, 19.2k, 28.8k, 38.4k, 57.6k, 115.2k or 230.4k. The following special externally clocked rates may be available, if the serial port supports them: MIDI\_1, MIDI\_16, MIDI\_32 and MIDI\_64. These correspond to rate multipliers of 1, 16, 32 and 64, respectively. Use of these special rates forces the data bits to 8 and the stop bits to 1. Note that the character ‘k’ is lower case.

**break**

Send a ‘break’ signal out the port of the serial device.

**chunk [on/off]**

Chunking of received data will be enabled (‘on’), disabled (‘off’) or reversed, if no argument is given. The chunk size is established when the *serialX* object is created.

**dtr [assert/negate/on/off]**

The DTR signal will asserted (‘assert’), negated (‘negate’) or used for handshaking (‘off’ or ‘on’ or no argument). When no argument is given, the use of the DTR signal for handshaking will be toggled. The DTR handshake is set active when the *serialX* object is created.

***float***

Send the single character that is the ASCII equivalent to the given value, out the port of the serial device.

***integer***

Send the single character that is the ASCII equivalent to the given value, out the port of the serial device.

**list anything**

Send the contents of the list out the port of the serial device. Integer and float values are sent as the equivalent ASCII character and symbols are sent as a string of ASCII characters. Note that no white space is introduced between values, so the command '**1 2 3 alpha 4**' results in the characters '123alpha4' being sent out the port of the serial device.

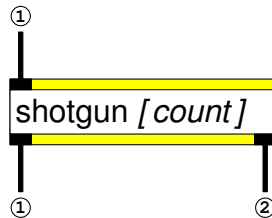
**text anything**

Send the characters corresponding to the arguments to the command out the port of the serial device. Note that no white space is introduced between values, so the command '**text 1 2 3 4**' results in the characters '1234' being sent out the port of the serial device.

**Comments:**

The *serialX* object was designed to address some weaknesses of the standard *serial* object—the 'break' signal was neither reported nor generated, the maximum baud rate was well below what most serial ports can handle, and output was always in the form of single characters. *serialX* was originally developed as a companion for the *gvp* object (which needed to be able to send a 'break' signal), and was extended to support higher baud rates and 'chunking' for the *mtc* object.

## *shotgun*



**Description of object:** *shotgun* redirects a bang to a randomly selected outlet.

**Object created:** July 2005

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Arith/Logic/Bitwise

**Argument(s):**

*num-outlets* (optional) integer, the number of outlets to select from

**Inlet(s):**

1 **bang**, the ‘bang’ to be sent out the selected outlet

**Outlet(s):**

1 **bang**, the first outlet

2 **bang**, the second outlet

⋮

*n* **bang**, the *n*th outlet

**Companion object(s):** none

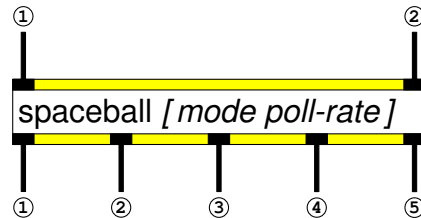
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

## *spaceball*



**Description of object:** *spaceball* is an interface to the [Labtec](#) six-degrees-of-freedom trackball, the Spaceball. It sends commands to a *serial* or a *serialX* object, which controls the serial port that the Spaceball is attached to, and responds to data returned from the Spaceball via the *serial* or *serialX* object.

**Object created:** July 2001

**Current version:** 1.0.3

**Online help file:** no

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):**

***mode* (optional) symbol**, the initial processing mode (additive ('add') or differential ('delta')) that is to be used. The default mode is additive.

***poll-rate* (optional) integer**, the rate (in milliseconds) at which the companion *serial* or *serialX* object is polled via a sample request. The default rate is 100 milliseconds between sample requests.

**Inlet(s):**

- 1 **list**, the command channel
- 2 **anything**, the output of the companion *serial* or *serialX* object

**Outlet(s):**

- 1 **list**, result data from the Spaceball, in the form of a button list (a three element list, starting with the symbol 'button', followed by the button number and the button state, where '0' is up and '1' is down, for each button transition), a rotation list (a four element list, starting with the symbol 'rotate', with the cumulative rotation factors as three floating point numbers, in the order 'X', 'Y', 'Z') or a translation list (a four element list, starting with the symbol 'translate', with the cumulative translation terms as three floating point numbers, in the order 'X', 'Y', 'Z'). Moving the sensor ball results in both a rotation list and a translation list; pressing a button results in two button lists, the first when the button is pressed and the second when it is released
- 2 **bang**, the sample request to send to the companion *serial* or *serialX* object
- 3 **anything**, the data to send to the companion *serial* or *serialX* object



**4 bang**, the ‘**chunk**’ request to send to the companion *serial* or *serialX* object, via a message object

**5 bang**, an error was detected

**Companion object(s):** works with *serialX* objects with OS 9, or *serial* objects with OS X

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**mode symbol**

Change the processing mode (where **symbol** is ‘add’ or ‘delta’) that is to be used.

**reset**

Send a device reset sequence to the Spaceball and reset the rotation and translation vectors.

**verbose [on/off]**

Tracing to the *Max* window of the messages sent will be enabled (‘on’), disabled (‘off’) or reversed, if no argument is given.

**zero**

Reset the rotation and translation vectors.

**Comments:**

Figure 20 shows how to connect a *spaceball* object to a *serial* object. Figure 21 shows how to connect a *spaceball* object to a *serialX* object.

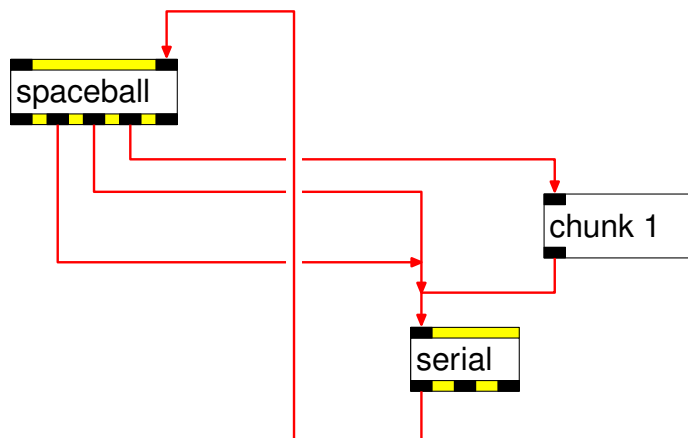


Figure 20: OS X: Connecting a *spaceball* object to a *serial* object

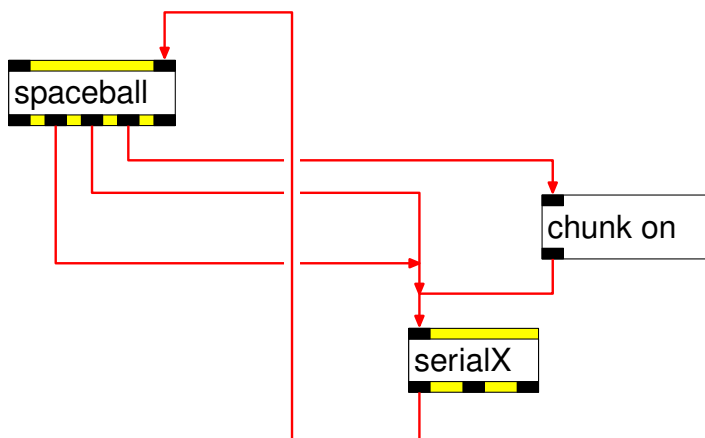
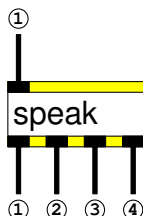


Figure 21: OS 9: Connecting a *spaceball* object to a *serialX* object

## *speak*



**Description of object:** *speak* is an interface to the Macintosh Speech Synthesis Manager, providing control over pitch, rate, volume and voice.

**Object created:** April 2001

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):** none

**Inlet(s):**

- 1 integer/float/list, the command input

**Outlet(s):**

- 1 **bang**, speaking has stopped
- 2 **bang**, speaking has paused
- 3 **list**, the response to the **'pitch?'**, **'rate?'**, **'voice?'**, **'voiceMax'** or **'volume?'** commands. Response messages appear as a two element list, starting with one of the symbols **'pitch'**, **'rate'**, **'voice'**, **'voicemax'**, **'volume'** and followed by the appropriate value—an integer value for **'voice'** and **'voicemax'** and a floating-point value for **'pitch'**, **'rate'** and **'volume'**.
- 4 **bang**, an error was detected

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the Speech Manager settings

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:****continue**

Cause the Speech Manager to resume speaking if it is paused.

**float**

Send the given number to the Speech Manager to be spoken immediately.

**integer**

Send the given number to the Speech Manager to be spoken immediately.

**list anything**

Send the given list to the Speech Manager to be spoken immediately.

**pause**

Cause the Speech Manager to pause speaking immediately.

**pitch float**

Select a new pitch for speaking. Higher values correspond to higher frequencies.

**pitch?**

Report the speech pitch as a two element list, with the symbol 'pitch' as its first element and the pitch as a floating-point number as its second element.

**rate float**

Select a new rate for speaking. The given value is measured (approximately) in terms of words-per-minute. Use the '**rate?**' command to get the current rate in a form that can be sent back to a *Speak* object.

**rate?**

Report the speech rate as a two element list, with the symbol 'rate' as its first element and the rate as a floating-point number as its second element.

**say anything**

Send the given list to the Speech Manager to be spoken immediately. This message permits the speaking of an arbitrary list; in particular any list that could be confused with a command to the *Speak* object.

**spell [on/off]**

Spelling of the following messages will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**stop**

Cause the Speech Manager to immediately stop speaking.

**voice integer**

Select a new voice to be used for speaking. Use the '**voice?**' command to get the current voice in a form that can be sent back to a *Speak* object.

**voice?**

Report the active voice number as a two element list, with the symbol 'voice' as its first element and the voice number as an integer as its second element.

**voicemax?**

Report the maximum voice number as a two element list, with the symbol 'voicemax' as its first element and the maximum voice number as an integer as its second element.

**volume float**

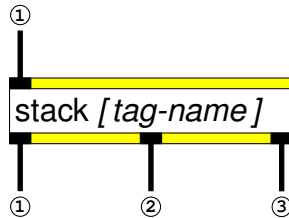
Select a new volume for speaking. The given value is between zero (silence) and one (maximum possible volume). Use the '**volume?**' command to get the current volume in a form that can be sent back to a *Speak* object.

object.

**volume?**

Report the speech volume as a two element list, with the symbol 'volume' as its first element and the volume as a floating-point number as its second element.

## *stack*



**Description of object:** *stack* is an implementation of push-down stacks, also known as LIFO (last-in, first-out) queues. Unlike conventional stacks, the *stack* object can store lists as well as single values on each level.

**Object created:** June 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Programming aids

**Object class(es):** Data

**Argument(s):**

*tag-name* (optional) **symbol**, share the internal stack with all other *stack* objects having the same **tag-name**

**Inlet(s):**

1 **list**, the command input

**Outlet(s):**

1 **anything**, the retrieved data

2 **integer**, the depth of the internal stack (the number of elements in the internal stack)

3 **bang**, an error was detected

**Companion object(s):** none

**Standalone:** no, the object works with other *stack* objects with the same **tag-name**

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**clear**

Remove all data from the internal stack.

**depth**

Return the depth of the internal stack.

**dup**

Duplicate the top element of the internal stack.

**pop**

Remove the top element of the internal stack.

**push anything**

Place the given data on the top of the internal stack.

**swap**

Exchange the top two elements of the internal stack.

**top**

Return the top element of the internal stack.

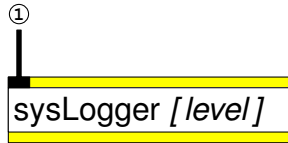
**top+pop**

Return the top element of the internal stack and remove it from the internal stack.

**trace [on/off]**

Internal stack update tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

## *sysLogger*



**Description of object:** *sysLogger* writes its input to the syslogd facility for Mac OS 9, available from [Brian Bergstrand](#) or to the native Mac OS X facility.

**Object created:** March 2002

**Current version:** 1.0.3

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Miscellaneous

**Argument(s):**

*level* (optional) **symbol**, the logging level to be used. The acceptable values are: 'emergency', 'alert', 'critical', 'error', 'warning', 'notice', 'info' and 'debug'. The default level is 'info'.

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):** none

**Companion object(s):** none

**Standalone:** yes

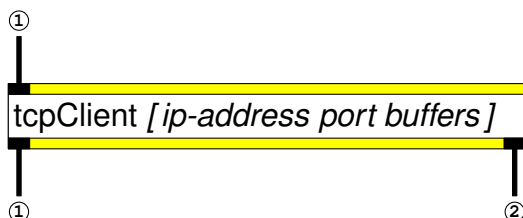
**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x*

**Fat, PPC-only or 68k-only:** Fat

**Comments:** Use of the *sysLogger* object in Mac OS X environment may require additions to the file `/etc/syslog.conf`; the output can be viewed using the standard Console application.



*tcpClient*

**Description of object:** *tcpClient* is an interface to the TCP/IP stack on a Macintosh, providing an end-point client to communicate with a *tcpServer* or a *tcpMultiServer* object.

**Object created:** August 1998

**Current version:** 1.1.7

**Online help file:** yes

**Object theme:** TCP/IP

**Object class(es):** Devices

**Argument(s):**

*ip-address* **(optional) symbol**, the address of the machine running the *tcpServer* or *tcpMultiServer* object to communicate with, in 'dotted' notation. The default address is '10.11.12.13'.

*port* **(optional) integer**, the number of the port to communicate on. The default port is 65535, which is also the maximum acceptable port number.

*buffers* **(optional) integer**, the number of receive buffers to use. The default number of buffers is 25

**Inlet(s):**

**1 list**, the command input

**Outlet(s):**

**1 list**, the status or response. Status messages (triggered by a '**bang**' or '**status**' command) appear as a two or four element list, starting with the symbol 'status'; response messages appear as a list, starting with the symbol 'reply' and 'self' messages appear as a two element list, starting with the symbol 'self'

**2 bang**, an error was detected

**Companion object(s):** none

**Standalone:** no, works with a *tcpServer* or a *tcpMultiServer* object on the same computer or another computer that is reachable via a TCP/IP network

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

***bang***

Report the state of the communication ('unbound', 'bound', 'listening', 'connecting', 'connected', 'disconnecting' or 'unknown') as a two or five element list, with the symbol 'status' as its first element. If the state is not 'connected', the list will contain two elements. If the state is 'connected', the third element is the IP address of the server, expressed as a symbol in 'dotted' notation, the fourth element is the port of the server and the fifth element is either 'raw' or 'max'.

**connect**

Connect to the *tcpServer* or *tcpMultiServer* object on the machine at the specified address and port.

**disconnect**

Close any existing connection to a *tcpServer* or a *tcpMultiServer* object.

***float***

Send the given floating point value to the *tcpServer* or *tcpMultiServer* object.

***integer***

Send the given integer value to the *tcpServer* or *tcpMultiServer* object.

**list anything**

Send the given list to the *tcpServer* or *tcpMultiServer* object.

**mode symbol**

Set the operating mode of the communication to either 'raw' or 'max'. In 'raw' mode, data is transferred without any structure; in 'max' mode it is as described below. Both ends of the communication must agree on the mode.

**port integer**

Set the number of the port to communicate on.

**self**

Returns the IP address of this object as a two element list, with the symbol 'self' as its first element.

**send anything**

Send the arguments to the *tcpServer* or *tcpMultiServer* object. This message permits the transmission of an arbitrary list; in particular any list that could be confused with a command to the *tcpClient* object.

**server symbol**

Specify the address of the machine running the *tcpServer* or *tcpMultiServer* object to communicate with, in 'dotted' notation.

**status**

Report the state of the communication ('unbound', 'bound', 'listening', 'connecting', 'connected', 'disconnecting' or 'unknown') as a two- or five element list, with the symbol 'status' as its first element. If the state is not 'connected', the list will contain two elements. If the state is 'connected', the third element is the IP address of the server, expressed as a symbol in 'dotted' notation, the fourth element is the port of the server and the fifth element is either 'raw' or 'max'.

**verbose [on/off]**

Communication tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**Message format:**

The data sent through the TCP/IP objects is structured as follows:

- 1) A fixed header containing the following fields:
  - a) The number of Atoms which follow, as a two byte number,
  - b) A sanity check field as a two byte number and
  - c) The type of Atoms to follow, as a single byte
- 2) Zero or more Atoms, represented via the following fields:
  - a) (optional) The type of the Atom, as a single byte,
  - b) The Atom, represented as described next.

All multi-byte values are sent most-significant-byte first (network byte order, which is big-endian). The sanity check field is the negative of the sum of the total number of bytes in the data and the number of Atoms sent. Floating-point values are represented using IEEE 32-bit floating point. Integers are represented as signed four byte quantities. Symbols are represented as strings, with a leading length (a two byte number) and a trailing null character. Note that the length includes the trailing null character, so it is always non-zero.

If all the Atoms are the same type, the type is placed in the header and the individual Atom type fields are not transmitted. If there is more than one type of Atom present, the Atom type field is transmitted for each Atom, and the type in the fixed header is set to eight (8). If the number of Atoms is zero, the type in the fixed header is also set to zero.

The type of the Atom is one of the following values:

- 1: Integer,
- 2: Floating point,
- 3: String,
- 10: Semicolon,
- 11: Comma or
- 12: Dollar

The following sequence of values represent the list '123 14.7, Chuck':

```
04      The number of Atoms which follow
-27     The sanity check field (3 Atoms, 24 bytes)
8       The type of Atoms (eight indicates a mixture of Atoms)
1       An integer value follows
0123    The value '123' as a 32-bit integer
2       A floating-point value follows
14.70   The value '14.7' as a 32-bit floating-point number
11      A comma follows (no associated data)
3       A string value follows
06      The length of the string
Chuck   The characters of the string
0       The trailing null character of the string
```

The largest message that can be processed is 24,000 bytes, which could contain 6,000 integers or floating-point values or a single string of 23,997 characters.

**Comments:**

Once a communication path is established between a *tcpClient* object and a *tcpServer* or *tcpMultiServer* object, either object can send messages—the path is full-duplex. Figure 22 is a state diagram for *tcpClient* objects.

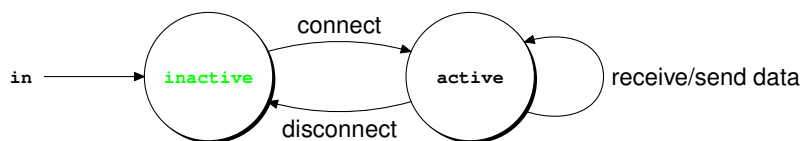
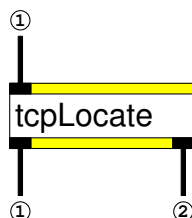


Figure 22: State diagram for *tcpClient* objects

## *tcpLocate*



**Description of object:** *tcpLocate* is an interface to the TCP/IP stack on a Macintosh, providing a client to identify the IP address corresponding to an Internet address.

**Object created:** November 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** TCP/IP

**Object class(es):** Devices

**Argument(s):** None

**Inlet(s):**

1 **list**, the command input

**Outlet(s):**

1 **list**, the response as a symbol

2 **bang**, an error was detected

**Companion object(s):** none

**Standalone:** no, works with a *tcpClient*, *tcpServer* or *tcpMultiServer* object

**Retains state:** yes

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

**symbol**

Interpret the given value as an Internet address and return the IP address if found.

**verbose [on/off]**

Communication tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

## *tcpMultiServer*



**Description of object:** *tcpMultiServer* is an interface to the TCP/IP stack on a Macintosh, providing an endpoint server to communicate with one or more *tcpClient* objects.

**Object created:** October 2000

**Current version:** 1.1.0

**Online help file:** yes

**Object theme:** TCP/IP

**Object class(es):** Devices

**Argument(s):**

**port (optional) integer**, the number of the port to communicate on. The default port is 65535, which is also the maximum acceptable port number

**clients (optional) integer**, the maximum number of clients that will be supported. The maximum that can be specified is 100 and the default is 5.

**buffers (optional) integer**, the total number of receive buffers to use. The default number of buffers is 25 times the maximum number of clients

**Inlet(s):**

**1 list**, the command input

**Outlet(s):**

**1 list**, the status or response. Status messages (triggered by a **'bang'** or **'status'** command) appear as a five element list, starting with the symbol **'status'**; response messages appear as a list, starting with the symbol **'reply'** and **'self'** messages appear as a two element list, starting with the symbol **'self'**. The second element of the status or response message is the identifier of the client connection involved.

**Companion object(s):** none

**Standalone:** no, works with multiple *tcpClient* objects on the same computer or other computers that are reachable via a TCP/IP network

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

***bang***

Report the general state of the communication as a five element list, with the symbol 'status' as its first element. The second element of the list is the value '0', the third element is the general state of the communication ('unbound', 'bound', 'listening' or 'unknown'), the fourth element is the number of active client connections and the fifth element is the port number that is being listened to.

**disconnect client**

Close any existing connection to the *tcpClient* object with the given identifier, if **client** is non-zero, or close all existing connections if **client** is zero.

**listen on/off**

Start listening on the communication port ('on') or stop listening ('off'). The command is only accepted if the *tcpMultiServer* object is in an appropriate state— 'listening' if 'off' and 'bound' if 'on'.

**mode client symbol**

Set the operating mode of the communication to either 'raw' or 'max'. In 'raw' mode, data is transferred without any structure; in 'max' mode it is as described below. Both ends of the communication must agree on the mode.

**port integer**

Set the number of the port to communicate on.

**self**

Returns the IP address of this object as a two element list, with the symbol 'self' as its first element.

**send client anything**

Send the arguments to the connected *tcpClient* objects with the given identifier, if **client** is non-zero, or send the arguments to all existing connections if **client** is zero (a broadcast). This message permits the transmission of an arbitrary list.

**status [client]**

Report either the state of a specific client communication, if **client** is non-zero (a valid identifier), or the general state of the communication, if **client** is zero, as a five or six element list, with the symbol 'status' as its first element. The second element of the list is **client**, the third element is the general state of the communication ('unbound', 'bound', 'listening' or 'unknown') or the specific client communication ('connecting', 'connected', 'disconnecting' or 'unknown'). If **client** is zero, the fourth element is the number of active client connections and the fifth element is the port number that is being listened to. If **client** is non-zero and the state is 'connected', the fourth element is the IP address of the client, expressed as a symbol in 'dotted' notation, the fifth element is the port of the client and the fifth element is either 'raw' or 'max'. The default for **client** is 0, which requests the general state of the communication.

**verbose [on/off]**

Communication tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**Message format:** see *tcpClient*

**Comments:**

Once a communication path is established between a *tcpClient* object and a *tcpMultiServer* object, either object can send messages—the path is full-duplex. The client identifier used with the commands ‘**disconnect**’, ‘**send**’ and ‘**status**’ and returned as part of the ‘reply’ and ‘status’ messages is a positive integer between 1 and the maximum number of clients specified when the *tcpMultiServer* object was created. Note that the identifier exists until the connection is closed with a ‘**disconnect**’ command (sent to the *tcpMultiServer* object or the connected *tcpClient* object) and the identifier may be reassigned to a subsequent connection. Note as well that only there can only be one *tcpMultiServer*, *tcpServer* or *udpPort* object for any given port. Figure 23 is a state diagram for *tcpMultiServer* objects.

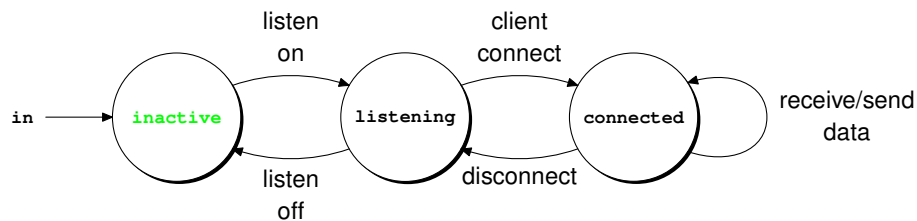
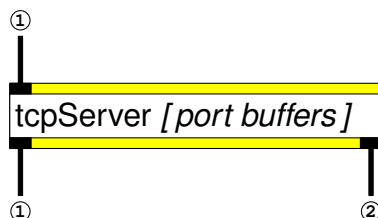


Figure 23: State diagram for *tcpMultiServer* objects



## *tcpServer*



**Description of object:** *tcpServer* is an interface to the TCP/IP stack on a Macintosh, providing an end-point server to communicate with a single *tcpClient* object.

**Object created:** August 1998

**Current version:** 1.1.7

**Online help file:** yes

**Object theme:** TCP/IP

**Object class(es):** Devices

**Argument(s):**

**port** (optional) **integer**, the number of the port to communicate on. The default port is 65535, which is also the maximum acceptable port number

**buffers** (optional) **integer**, the number of receive buffers to use. The default number of buffers is 25

**Inlet(s):**

**1 list**, the command input

**Outlet(s):**

**1 list**, the status or response. Status messages (triggered by a '**bang**' or '**status**' command) appear as a three or four element list, starting with the symbol 'status'; response messages appear as a list, starting with the symbol 'reply' and 'self' messages appear as a two element list, starting with the symbol 'self'.

**2 bang**, an error was detected

**Companion object(s):** none

**Standalone:** no, works with a *tcpClient* object on the same computer or another computer that is reachable via a TCP/IP network

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:*****bang***

Report the state of the communication ('unbound', 'bound', 'listening', 'connecting', 'connected', 'disconnecting' or 'unknown') as a three or five element list, with the symbol 'status' as its first element. If the state is not 'connected', the third element of the list is the port number that is being listened on. If the state is 'connected', the third element is the IP address of the client, expressed as a symbol in 'dotted' notation, the fourth element is the port of the server and the fifth element is either 'raw' or 'max'.

**disconnect**

Close any existing connection to a *tcpClient* object.

***float***

Send the given floating point value to the *tcpClient* object.

***integer***

Send the given integer value to the *tcpClient* object.

**list anything**

Send the given list to the *tcpClient* object.

**listen on/off**

Start listening on the communication port ('on') or stop listening ('off'). The command is only accepted if the *tcpServer* object is in an appropriate state— 'listening' if 'off' and 'bound' if 'on'.

**mode symbol**

Set the operating mode of the communication to either 'raw' or 'max'. In 'raw' mode, data is transferred without any structure; in 'max' mode it is as described below. Both ends of the communication must agree on the mode.

**port integer**

Set the number of the port to communicate on.

**self**

Returns the IP address of this object as a two element list, with the symbol 'self' as its first element.

**send anything**

Send the arguments to the *tcpClient* object. This message permits the transmission of an arbitrary list; in particular any list that could be confused with a command to the *tcpServer* object.

**status**

Report the state of the communication ('unbound', 'bound', 'listening', 'connecting', 'connected', 'disconnecting' or 'unknown') as a two or five element list, with the symbol 'status' as its first element. If the state is not 'connected', the list will contain two elements. If the state is 'connected', the third element is the IP address of the client, expressed as a symbol in 'dotted' notation, the fourth element is the port of the client and the fifth element is either 'raw' or 'max'.

**verbose [on/off]**

Communication tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

**Message format:**                      see *tcpClient*

**Comments:**

Once a communication path is established between a *tcpClient* object and a *tcpServer* object, either object can send messages—the path is full-duplex. Note as well that only there can only be one *tcpMultiServer*, *tcpServer* or *udpPort* object for any given port. Figure 24 is a state diagram for *tcpServer* objects.

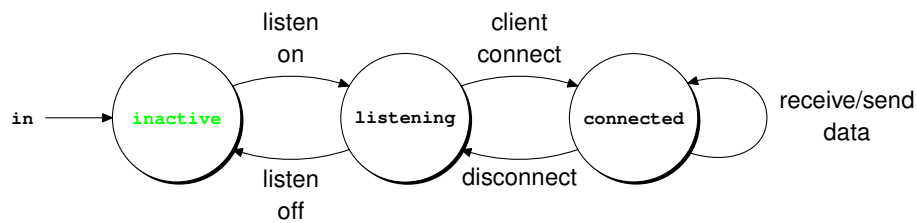
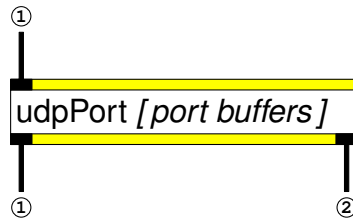


Figure 24: State diagram for *tcpServer* objects

## *udpPort*



**Description of object:** *udpPort* is an interface to the UDP/IP stack on a Macintosh, providing an endpoint to communicate with another *udpPort* object.

**Object created:** July 2005

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** TCP/IP

**Object class(es):** Devices

**Argument(s):**

**port** (optional) integer, the number of the port to communicate on. The default port is 65535, which is also the maximum acceptable port number

**buffers** (optional) integer, the number of receive buffers to use. The default number of buffers is 25

**Inlet(s):**

**1 list**, the command input

**Outlet(s):**

**1 list**, the status or response. Status messages (triggered by a **'bang'** or **'status'** command) appear as a three or four element list, starting with the symbol **'status'**; response messages appear as a list, starting with the symbol **'reply'** and **'self'** messages appear as a two element list, starting with the symbol **'self'**.

**2 bang**, an error was detected

**Companion object(s):** none

**Standalone:** no, works with another *udpPort* object on the same computer or another computer that is reachable via a TCP/IP network

**Retains state:** yes

**Compatibility:** Max 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:*****bang***

Report the state of the communication ('unbound', 'bound' or 'unknown') as a three element list, with the symbol 'status' as its first element. The third element is either 'raw' or 'max'.

***float***

Send the given floating point value to the other *udpPort* object.

***integer***

Send the given integer value to the other *udpPort* object.

**list anything**

Send the given list to the other *udpPort* object.

**mode symbol**

Set the operating mode of the communication to either 'raw' or 'max'. In 'raw' mode, data is transferred without any structure; in 'max' mode it is as described below. Both ends of the communication must agree on the mode.

**port integer**

Set the number of the port to communicate on.

**self**

Returns the IP address of this object as a two element list, with the symbol 'self' as its first element.

**send anything**

Send the arguments to the other *udpPortX* object. This message permits the transmission of an arbitrary list; in particular any list that could be confused with a command to the *udpPort* object.

**sendTo symbol integer**

Specify the address of the machine running the other *udpPort* object to communicate with, in 'dotted' notation, along with the port to communicate on.

**status**

Report the state of the communication ('unbound', 'bound' or 'unknown') as a three element list, with the symbol 'status' as its first element. The third element is either 'raw' or 'max'.

**verbose [on/off]**

Communication tracing to the *Max* window will be enabled ('on'), disabled ('off') or reversed, if no argument is given.

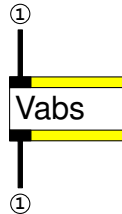
**Message format:**

see *tcpClient*

**Comments:**

Once a communication path is established between a *udpPort* object and another *udpPort* object, either object can send messages—the path is full-duplex. Note as well that only there can only be one *tcpMultiServer*, *tcpServer* or *udpPort* object for any given port.

## Vabs



**Description of object:** *Vabs* calculates the absolute value of the input (either a list or a single number).

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):**

**1 anything**, the input after calculating the absolute value, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

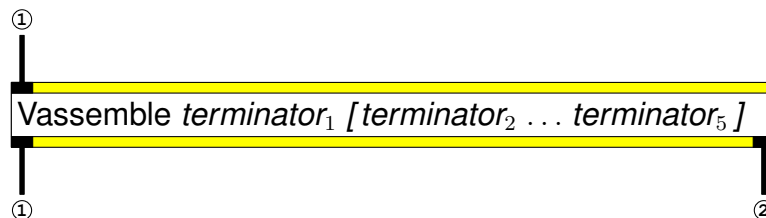
**Command language syntax:**

*bang*

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow |x_i|$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.

## Vassemble



**Description of object:** *Vassemble* is used to collect a sequence of numbers that are terminated by one of a set of numbers. The ‘terminator’ numbers are removed from the sequence.

**Object created:** June 2003

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*terminator1* **integer**, a number that marks the end-of-list. Only non-zero numbers will be recognized.

*terminator2* **(optional) integer**, another number that marks the end-of-list.

*terminator3* **(optional) integer**, another number that marks the end-of-list.

*terminator4* **(optional) integer**, another number that marks the end-of-list.

*terminator5* **(optional) integer**, another number that marks the end-of-list.

**Inlet(s):**

**1 integer/list/bang**, the list to be processed. A single number is treated as a single element list.

**Outlet(s):**

**1 list**, the accumulated list

**2 bang**, an empty list was generated

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the terminator numbers and the numbers accumulated so far

**Compatibility:** *Max 4.x* {OS 9 and OS X}

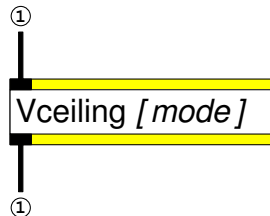
**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

*bang*

Return the previous result, if any.

## Vceiling



**Description of object:** *Vceiling* calculates the smallest integer greater than the value given (either a list or a single number).

**Object created:** November 2000

**Current version:** 1.0.5

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*mode* (**optional symbol**, either 'f', 'i' or 'm' to indicate whether the output is to be floating-point values only, integer values only, or mixed values. Mixed values are floating-point if the input is floating-point and integer if the input is integer. The default is 'm'.

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):**

**1 anything**, the input after calculating the ceiling, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

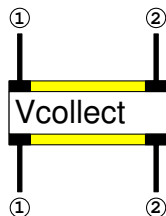
*bang*

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow \lceil x_i \rceil$ , where  $x_1, x_2, \dots, x_n$  is the inlet value and  $y_1, y_2, \dots, y_n$  is the outlet result.



## *Vcollect*



**Description of object:** *Vcollect* collects all the data arriving at its left inlet into a list.

**Object created:** August 2002

**Current version:** 1.0.1

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):** none

**Inlet(s):**

- 1 anything**, the data to be collected
- 2 symbol**, the command input

**Outlet(s):**

- 1 list**, the collected data
- 2 integer**, the count of the atoms collected

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

***bang***

Output the collected data.

***clear***

Remove all the collected data.

**count**

Return the number of atoms collected so far.

**start**

Start collecting data.

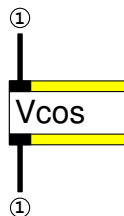
**stop**

Stop collecting data.

**Comments:**

The *Vcollect* object was designed to address some problems that were found in attempting to use *coll* objects to gather arbitrary sequential streams of data.

## Vcos



**Description of object:** *Vcos* calculates the cosine of the input (either a list or a single number).

**Object created:** May 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

**1** **anything**, the input to be processed

**Outlet(s):**

**1** **anything**, the input after calculating the cosine, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

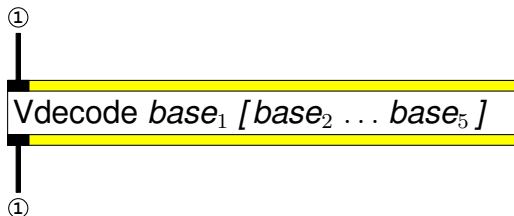
**Command language syntax:**

***bang***

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow \cos x_i$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.

## Vdecode



**Description of object:** *Vdecode* is an implementation of the *APL* ‘decode’ operator ( $\perp$ ), which is used to convert a coded representation of a number into the number itself.

**Object created:** June 2003

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

**base1 integer**, the leftmost base to be applied. If this is the only base it will be repeated as often as needed to do the conversion. If there are more bases and this base is negative, it will be repeated (as its absolute value) as often as needed to do the conversion, after the other bases have been applied. Bases must be greater than 1.

**base2 (optional) integer**, the next-to-last base to apply.

**base3 (optional) integer**, the next-to-next-to-last base to apply.

**base4 (optional) integer**, the next-to-...-last base to apply.

**base5 (optional) integer**, the last base to apply. The elements of the input list are multiplied by each base, in sequence, and the results are summed into the output number.

**Inlet(s):**

**1 list/number**, the list to be decoded. A single number is treated as a single element list.

**Outlet(s):**

**1 number**, the decoded result

**Companion object(s):** none

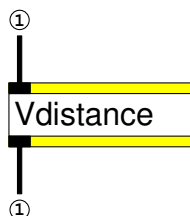
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** PPC-only

## *Vdistance*



**Description of object:** *Vdistance* calculates the length of its input list, considered as an n-dimensional vector.

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):** none

**Inlet(s):**

**1** anything, the input to be processed

**Outlet(s):**

**1** anything, the input after calculating the length, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

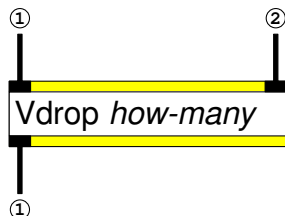
**Command language syntax:**

***bang***

Return the previous result, if any.

**Comments:** In mathematical terms:  $y \leftarrow |x|$  or  $y \leftarrow \sqrt{(\sum_{i=1}^n x_i^2)}$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.

## Vdrop



**Description of object:** *Vdrop* is an implementation of the *APL* ‘drop’ operator ( $\downarrow$ ), which is used to return the remainder of a vector (in *Max*, a list) with leading or trailing elements removed.

**Object created:** July 2000

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):**

*how-many* **integer**, the number of elements to drop. A positive number indicates that the elements are removed from the beginning of the input list, while a negative number indicates that the elements are to be removed from the end of the list.

**Inlet(s):**

**1 bang/list**, the list to be reduced

**2 integer**, the number of elements to drop. This replaces the initial argument.

**Outlet(s):**

**1 list**, the reduced list, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the number of elements to drop

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

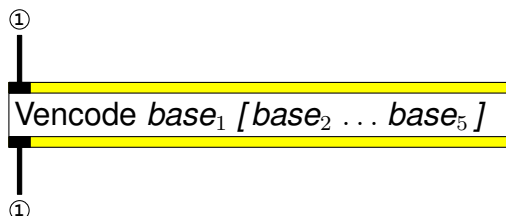
**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

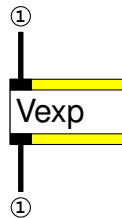
Return the previous result, if any.

## Vencode



<b>Description of object:</b>	<i>Vencode</i> is an implementation of the <i>APL</i> ‘encode’ operator (⍒), which is used to convert a number into an encoded representation according to a coding scheme or base.
<b>Object created:</b>	June 2003
<b>Current version:</b>	1.0.0
<b>Online help file:</b>	yes
<b>Object theme:</b>	Vector manipulation
<b>Object class(es):</b>	Arith/Logic/Bitwise, Lists
<b>Argument(s):</b>	<p><i>base1</i> <b>integer</b>, the leftmost base to be applied. If this is the only base it will be repeated as often as needed to do the conversion. If there are more bases and this base is negative, it will be repeated (as its absolute value) as often as needed to do the conversion, after the other bases have been applied. Bases must be greater than 1.</p> <p><i>base2</i> <b>(optional) integer</b>, the next-to-last base to apply.</p> <p><i>base3</i> <b>(optional) integer</b>, the next-to-next-to-last base to apply.</p> <p><i>base4</i> <b>(optional) integer</b>, the next-to-...-last base to apply.</p> <p><i>base5</i> <b>(optional) integer</b>, the last base to apply. The input number is divided by each base, in sequence, and the remainders are collected into the output list.</p>
<b>Inlet(s):</b>	<p><b>1 list/number</b>, the number to be encoded. A list results in a sequence of lists being generated.</p>
<b>Outlet(s):</b>	<p><b>1 list</b>, the encoded result</p>
<b>Companion object(s):</b>	none
<b>Standalone:</b>	yes
<b>Retains state:</b>	no
<b>Compatibility:</b>	Max 4.x {OS 9 and OS X}
<b>Fat, PPC-only or 68k-only:</b>	PPC-only

## *Vexp*



**Description of object:** *Vexp* calculates the natural exponential of the input (either a list or a single number).

**Object created:** May 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):**

**1 anything**, the input after calculating the natural exponential, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

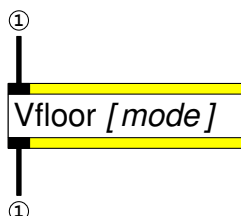
*bang*

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow e^{x_i}$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.



## *Vfloor*



**Description of object:** *Vfloor* calculates the largest integer less than the value given (either a list or a single number).

**Object created:** November 2000

**Current version:** 1.0.5

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*mode* (optional) symbol, either 'f', 'i' or 'm' to indicate whether the output is to be floating-point values only, integer values only, or mixed values. Mixed values are floating-point if the input is floating-point and integer if the input is integer. The default is 'm'.

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):**

**1 anything**, the input after calculating the floor, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

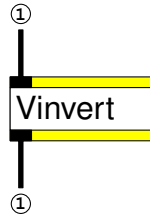
**Command language syntax:**

*bang*

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow \lfloor x_i \rfloor$ , where  $x_1, x_2, \dots, x_n$  is the inlet value and  $y_1, y_2, \dots, y_n$  is the outlet result.

## *Vinvert*



**Description of object:** *Vinvert* calculates the multiplicative inverse of the input (either a list or a single number).

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

**1** anything, the input to be processed

**Outlet(s):**

**1** anything, the input after calculating the multiplicative, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

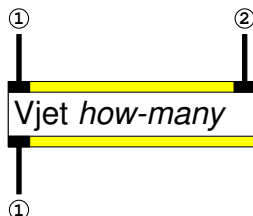
**Command language syntax:**

*bang*

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow \frac{1}{x_i}$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.

## Vjet



**Description of object:** *Vjet* takes as input a list and divides it into a series of fixed-size, shorter, lists. It is similar to the *APL* 'reshape' operator ( $\rho$ ).

**Object created:** July 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):**

*how-many* **integer**, the size of the output fragments

**Inlet(s):**

**1** **bang/list**, the input to be processed

**2** **integer**, the size of the output fragments. This replaces the initial argument.

**Outlet(s):**

**1** **list**, the input after segmentation, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the size of the output fragments

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

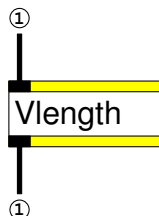
**Command language syntax:**

*bang*

Return the previous result, if any.

**Comments:** The *Vjet* object was designed to assist in working with the *mtc* object, which returns triples of values in the form of a long list. If the input list is not exactly divisible by the size of the output fragments, the last fragment will be shorter than all the earlier fragments.

## *Vlength*



**Description of object:** *Vlength* returns the number of elements in the list that it receives.

**Object created:** July 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):** none

**Inlet(s):**

1 list, the list to measure

**Outlet(s):**

1 integer, the size of the input

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## Vlog



**Description of object:** *Vlog* calculates the natural logarithm of the input (either a list or a single number).

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

**1** *anything*, the input to be processed

**Outlet(s):**

**1** *anything*, the input after calculating the natural logarithm, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

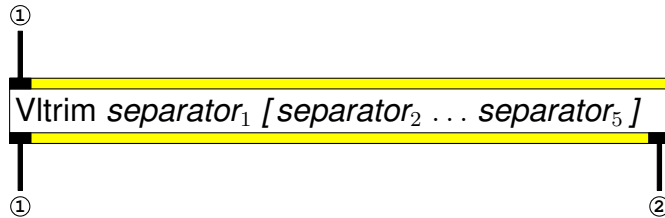
**Command language syntax:**

***bang***

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow \log_e x_i$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.

## *Vltrim*



**Description of object:** *Vltrim* is used to remove ‘noise’ numbers from the beginning of a list.

**Object created:** June 2003

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*separator1* **integer**, a ‘noise’ number to remove. Only non-zero numbers will be recognized.

*separator2* **(optional) integer**, another ‘noise’ number to remove.

*separator3* **(optional) integer**, another ‘noise’ number to remove.

*separator4* **(optional) integer**, another ‘noise’ number to remove.

*separator5* **(optional) integer**, another ‘noise’ number to remove.

**Inlet(s):**

**1 integer/list/bang**, the list to be processed. A single number is treated as a single element list.

**Outlet(s):**

**1 list**, the reduced list

**2 bang**, an empty list was generated

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the separator numbers

**Compatibility:** *Max 4.x* {OS 9 and OS X}

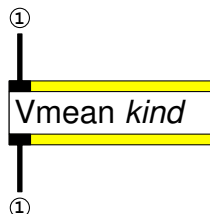
**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

*bang*

Return the previous result, if any.

## *Vmean*



<b>Description of object:</b>	<i>Vmean</i> calculates an arithmetic, geometric, or harmonic mean of the elements of a vector (in <i>Max</i> , a list).
<b>Object created:</b>	April 2001
<b>Current version:</b>	1.0.2
<b>Online help file:</b>	yes
<b>Object theme:</b>	Vector manipulation
<b>Object class(es):</b>	Lists
<b>Argument(s):</b>	<p><b><i>kind</i> symbol</b>, the kind of mean to be calculated. Recognized kinds are arithmetic ('a' or 'arith'), geometric ('g' or 'geom') and harmonic ('h' or 'harm').</p>
<b>Inlet(s):</b>	<p><b>1 list</b>, the list to be reduced</p>
<b>Outlet(s):</b>	<p><b>1 number</b>, the result of the calculation</p>
<b>Companion object(s):</b>	none
<b>Standalone:</b>	yes
<b>Retains state:</b>	no
<b>Compatibility:</b>	<i>Max</i> 3.x and <i>Max</i> 4.x {OS 9 and OS X}
<b>Fat, PPC-only or 68k-only:</b>	Fat
<b>Comments:</b>	Figure 25 is the definition of the output of the <i>Vmean</i> object in mathematical terms

$$y = \frac{\sum_{i=1}^n x_i}{n} \quad \text{arithmetic mean}$$

$$y = \sqrt[n]{\prod_{i=1}^n x_i} \quad \text{geometric mean}$$

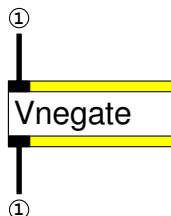
$$y = \frac{n}{\sum_{i=1}^n \frac{1}{x_i}} \quad \text{harmonic mean}$$

Where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y$  is the outlet result.

Figure 25: Definition of the output of *Vmean*



## *Vnegate*



**Description of object:** *Vnegate* calculates the negative value of the input (either a list or a single number).

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

1 **anything**, the input to be processed

**Outlet(s):**

1 **anything**, the input after calculating the negative value, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

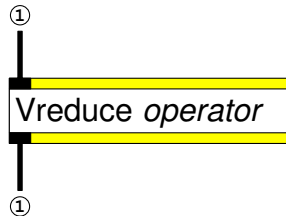
**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

Return the previous result, if any.

## Vreduce



**Description of object:** *Vreduce* is an implementation of the *APL* ‘reduction’ operator ( $f/$ ), which is used to apply an operator ( $f$ ) over the elements of a vector (in *Max*, a list).

**Object created:** November 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

**operator symbol**, the operator to be applied. Recognized operators are sum ( $+$ ), logical AND ( $\&\&$ ), bit-wise AND ( $\&$ ), bit-wise OR ( $|$ ), divide ( $/$ ), exclusive-OR ( $\wedge$ ), maximum ( $\text{max}$ ), minimum ( $\text{min}$ ), modulus ( $\%$ ), product ( $*$ ), logical OR ( $||$ ), or subtract ( $-$ ). The result of applying the operator to each element in order (with the previous result being used as the left operand and the new element being used as the right operand) is returned. For an empty list, the identity element (if defined for the operator), or zero, is returned.

**Inlet(s):**

**1 list/number**, the list to be reduced. A single number is treated as a single element list.

**Outlet(s):**

**1 number**, the reduction result

**Companion object(s):** none

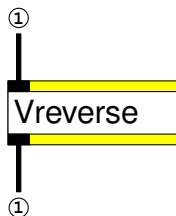
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## *Vreverse*



**Description of object:** *Vreverse* is an implementation of the *APL* ‘reverse’ operator (monadic  $\Phi$ ), which is used to reverse the order of the elements of a vector (in *Max*, a list).

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):** none

**Inlet(s):**

1 **bang**/list, the list to be reversed

**Outlet(s):**

1 **list**, the reversed list, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

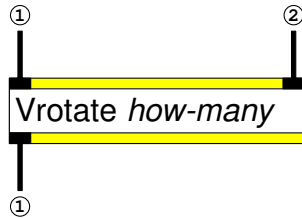
**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

Return the previous result, if any.

## *Vrotate*



**Description of object:** *Vrotate* is an implementation of the *APL* ‘rotate’ operator (dyadic  $\Phi$ ), which is used to rotate the elements of a vector (in *Max*, a list).

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):**

*how-many* **integer**, the number of elements to rotate. A positive number indicates that the beginning of the input list is moved towards the end of the input list by the number of elements given, while a negative number indicates that the end of the input list is moved towards the beginning of the input list by (the absolute value of) the number of elements given.

**Inlet(s):**

- 1 **bang/list**, the list to be rotated
- 2 **integer**, the number of elements to rotate. This replaces the initial argument.

**Outlet(s):**

- 1 **list**, the rotated list, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the number of elements to rotate

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

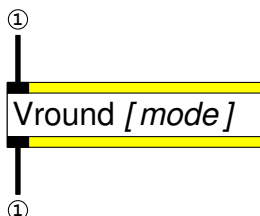
**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

Return the previous result, if any.

## Vround



**Description of object:** *Vround* calculates the integer nearest to the value given (either a list or a single number).

**Object created:** November 2000

**Current version:** 1.0.5

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*mode* (**optional**) **symbol**, either 'f', 'i' or 'm' to indicate whether the output is to be floating-point values only, integer values only, or mixed values. Mixed values are floating-point if the input is floating-point and integer if the input is integer. The default is 'm'.

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):**

**1 anything**, the input after calculating the nearest integer, or the previous result (if a 'bang' is received)

**Companion object(s):** none

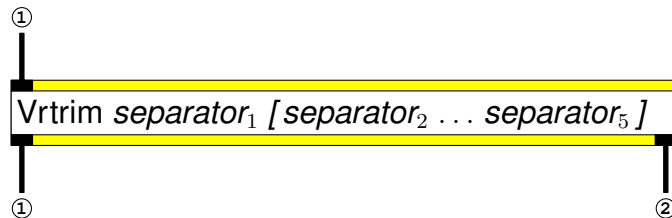
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## Vrtrim



**Description of object:** *Vrtrim* is used to remove ‘noise’ numbers from the end of a list.

**Object created:** June 2003

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*separator1* **integer**, a ‘noise’ number to remove. Only non-zero numbers will be recognized.

*separator2* **(optional) integer**, another ‘noise’ number to remove.

*separator3* **(optional) integer**, another ‘noise’ number to remove.

*separator4* **(optional) integer**, another ‘noise’ number to remove.

*separator5* **(optional) integer**, another ‘noise’ number to remove.

**Inlet(s):**

**1 integer/list/bang**, the list to be processed. A single number is treated as a single element list.

**Outlet(s):**

**1 list**, the reduced list

**2 bang**, an empty list was generated

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the separator numbers

**Compatibility:** *Max 4.x* {OS 9 and OS X}

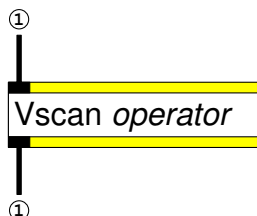
**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

*bang*

Return the previous result, if any.

## Vscan



**Description of object:** *Vscan* is an implementation of the *APL* 'scan' operator ( $f\backslash$ ), which is used to apply an operator ( $f$ ) over the elements of a vector (in *Max*, a list).

**Object created:** November 2000

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

**operator symbol**, the operator to be applied. Recognized operators are sum ('+'), logical AND ('&&'), bit-wise AND ('&'), bit-wise OR ('|'), divide ('/'), exclusive-OR ('^'), maximum ('max'), minimum ('min'), modulus ('%'), product ('\*'), logical OR ('||'), or subtract ('-'). The result of applying the operator to each element in order (with the previous result being used as the left operand and the new element being used as the right operand) are collected and returned as a new list. For an empty list, the identity element (if defined for the operator), or zero, is returned.

**Inlet(s):**

**1 list/number**, the list to be scanned. A single number is treated as a single element list.

**Outlet(s):**

**1 list**, the scan result

**Companion object(s):** none

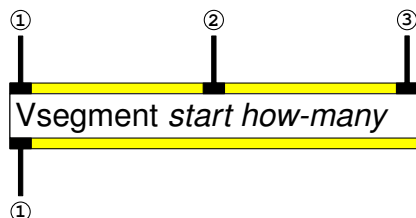
**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

## Vsegment



**Description of object:** *Vsegment* is used to extract a portion of a list.

**Object created:** July 2000

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):**

**start integer**, the starting element to select from the list. A positive number indicates that the selection starts from the beginning of the list, while a negative number indicates that the selection starts from the end of the list, with the last element having the position of ‘-1’.

**how-many integer**, the number of elements to select from the list. A positive number indicates that the selection extends from the starting element towards the end of the list, while a negative number indicates that the selection extends from the starting element towards the beginning of the list.

**Inlet(s):**

- 1 **bang/list**, the list to be reduced
- 2 **integer**, the starting element to select from the list. This replaces the initial argument **start**.
- 3 **integer**, the number of elements to select from the list. This replaces the initial argument **how-many**.

**Outlet(s):**

- 1 **list**, the reduced list, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the starting element and the number of elements

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

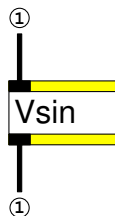
**Command language syntax:**

*bang*

Return the previous result, if any.



## Vsin



**Description of object:** *Vsin* calculates the sine of the input (either a list or a single number).

**Object created:** May 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):**

**1 anything**, the input after calculating the sine, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

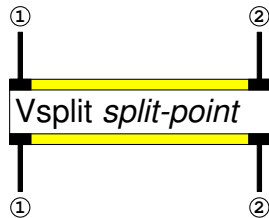
**Command language syntax:**

*bang*

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow \sin x_i$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.

## Vsplit



**Description of object:** *Vsplit* is a combination of a *Vtake* and *Vdrop*.

**Object created:** July 2005

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):**

*how-many* **integer**, the number of elements to split by. A positive number indicates that the split point is counted from the beginning of the input list, while a negative number indicates that the split point is counted from the end of the list.

**Inlet(s):**

- 1 **bang/list**, the list to be split
- 2 **integer**, the number of elements to split. This replaces the initial argument.

**Outlet(s):**

- 1 **list**, the left part of the list, or the previous result (if a 'bang' is received)
- 1 **list**, the right part of the list, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the number of elements to split

**Compatibility:** *Max 4.x* {OS 9 and OS X}

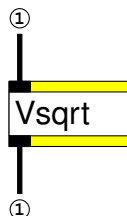
**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

*bang*

Return the previous result, if any.

## *Vsqr*



**Description of object:** *Vsqr* calculates the square root of the input (either a list or a single number).

**Object created:** April 2001

**Current version:** 1.0.2

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):** none

**Inlet(s):**

**1** anything, the input to be processed

**Outlet(s):**

**1** anything, the input after calculating the square root, or the previous result (if a 'bang' is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

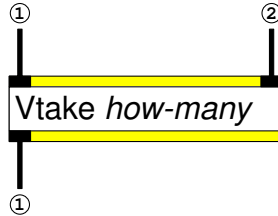
**Command language syntax:**

***bang***

Return the previous result, if any.

**Comments:** In mathematical terms:  $y_i \leftarrow \sqrt{x_i}$ , where  $x = x_1, x_2, \dots, x_n$  is the inlet value and  $y = y_1, y_2, \dots, y_n$  is the outlet result.

## Vtake



**Description of object:** *Vtake* is an implementation of the *APL* ‘take’ operator ( $\uparrow$ ), which is used to return leading or trailing elements of a vector (in *Max* terms, a list).

**Object created:** July 2000

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Lists

**Argument(s):**

*how-many* **integer**, the number of elements to take. A positive number indicates that the elements are taken from the beginning of the input list, while a negative number indicates that the elements are to be taken from the end of the list.

**Inlet(s):**

- 1 **bang/list**, the list to reduce
- 2 **integer**, the number of elements to take. This replaces the initial argument.

**Outlet(s):**

- 1 **list**, the reduced list, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the number of elements to take

**Compatibility:** *Max* 3.x and *Max* 4.x {OS 9 and OS X}

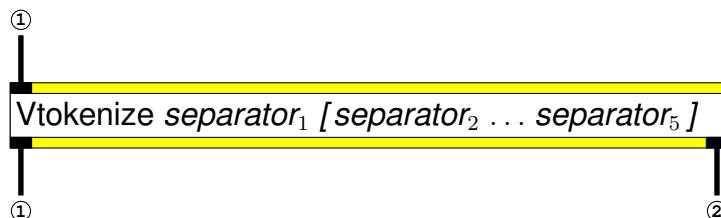
**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

Return the previous result, if any.

## Vtokenize



**Description of object:** *Vtokenize* is used to partition a list of numbers into a sequence of sublists, separated by ‘noise’ numbers in the original list.

**Object created:** June 2003

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*separator1* **integer**, a number that indicates the end of a sublist. Only non-zero numbers will be recognized.

*separator2* **(optional) integer**, another number that indicates the end of a sublist.

*separator3* **(optional) integer**, another number that indicates the end of a sublist.

*separator4* **(optional) integer**, another number that indicates the end of a sublist.

*separator5* **(optional) integer**, another number that indicates the end of a sublist.

**Inlet(s):**

**1 integer/list/bang**, the list to be processed. A single number is treated as a single element list.

**Outlet(s):**

**1 list**, the generated sublists

**2 bang**, the last sublist was generated

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the separator numbers

**Compatibility:** *Max 4.x* {OS 9 and OS X}

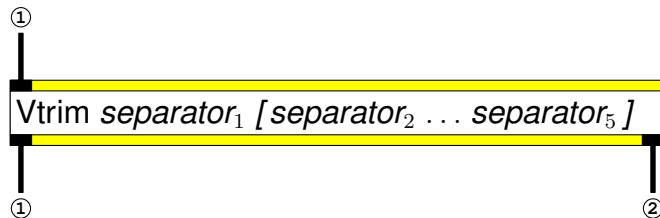
**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

*bang*

Return the previous sequence of sublists, if any.

## Vtrim



**Description of object:** *Vtrim* is used to remove ‘noise’ numbers from the beginning and end of a list.

**Object created:** June 2003

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*separator1* **integer**, a ‘noise’ number to remove. Only non-zero numbers will be recognized.

*separator2* **(optional) integer**, another ‘noise’ number to remove.

*separator3* **(optional) integer**, another ‘noise’ number to remove.

*separator4* **(optional) integer**, another ‘noise’ number to remove.

*separator5* **(optional) integer**, another ‘noise’ number to remove.

**Inlet(s):**

**1 integer/list/bang**, the list to be processed. A single number is treated as a single element list.

**Outlet(s):**

**1 list**, the reduced list

**2 bang**, an empty list was generated

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the separator numbers

**Compatibility:** *Max 4.x* {OS 9 and OS X}

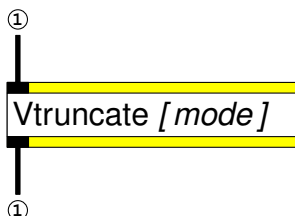
**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

*bang*

Return the previous result, if any.

## *Vtruncate*



**Description of object:** *Vtruncate* calculates the integer part of the value given (either a list or a single number).

**Object created:** November 2000

**Current version:** 1.0.5

**Online help file:** yes

**Object theme:** Miscellaneous

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*mode* (**optional**) **symbol**, either ‘f’, ‘i’ or ‘m’ to indicate whether the output is to be floating-point values only, integer values only, or mixed values. Mixed values are floating-point if the input is floating-point and integer if the input is integer. The default is ‘m’.

**Inlet(s):**

**1 anything**, the input to be processed

**Outlet(s):**

**1 anything**, the input after removing the fractional part, or the previous result (if a ‘bang’ is received)

**Companion object(s):** none

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

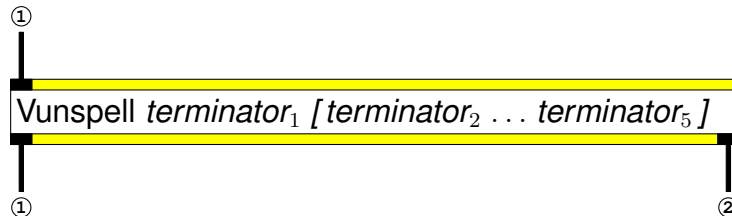
**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

Return the previous result, if any.

## Vunspell



**Description of object:** *Vunspell* is used to convert a sequence of numbers, representing ASCII characters, into the *Max* objects that they represent.

**Object created:** June 2003

**Current version:** 1.0.0

**Online help file:** yes

**Object theme:** Vector manipulation

**Object class(es):** Arith/Logic/Bitwise, Lists

**Argument(s):**

*terminator1* **integer**, a number that marks the end-of-list. Only non-zero numbers will be recognized.

*terminator2* **(optional) integer**, another number that marks the end-of-list.

*terminator3* **(optional) integer**, another number that marks the end-of-list.

*terminator4* **(optional) integer**, another number that marks the end-of-list.

*terminator5* **(optional) integer**, another number that marks the end-of-list.

**Inlet(s):**

**1 integer/list/bang**, the list to be processed. A single number is treated as a single element list.

**Outlet(s):**

**1 list**, the generated list of atoms

**2 bang**, an empty list was generated

**Companion object(s):** none

**Standalone:** yes

**Retains state:** yes, the terminator numbers and the numbers accumulated so far

**Compatibility:** *Max* 4.x {OS 9 and OS X}

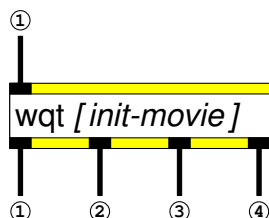
**Fat, PPC-only or 68k-only:** PPC-only

**Command language syntax:**

*bang*

Return the previous result, if any.



***wqt***

**Description of object:** *wqt* provides a windowed interface to QuickTime™ movies, permitting control of playback rate and the section of the movie to be played. It provides more functionality than the standard *movie* object.

**Object created:** October 1998

**Current version:** 1.0.4

**Online help file:** yes

**Object theme:** QuickTime™

**Object class(es):** Graphics

**Argument(s):**

*init-movie* (optional) symbol, the initial movie to be loaded

**Inlet(s):**

1 list, the command input

**Outlet(s):**

- 1 integer, the result of a command
- 2 integer, the duration of the current movie
- 3 bang, playing has stopped
- 4 bang, an error was detected

**Companion object(s):** yes, (optional) the standard *movie* play controller interface object can be attached to a *wqt* object

**Standalone:** yes

**Retains state:** yes

**Compatibility:** Max 3.x and Max 4.x {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:****active [0/1]**

Set the current movie active ('1') or inactive ('0').

***bang***

Start the current movie playing.

**begin**

Move to the beginning of the current movie and make it active.

**count**

Return the number of movies loaded.

**duration**

Return the length of the current movie.

**end**

Move to the end of the current movie and make it active.

**getrate**

Return the rate at which the current movie will be played.

**getvolume**

Return the audio level for the current movie.

***integer***

Move to the given frame number in the current movie.

**load [movie-name]**

Add the specified movie to the list of movies and make it the current movie. **movie-name** must be a symbol, not a number.

**mute [0/1]**

Change the audio level of the current movie, silencing it ('0') or restoring the previous level ('1').

**pause**

Stop the current movie.

**rate integer [integer]**

Set the rate at which the current movie will be played, using the ratio of the first number to the second, and start the movie playing. If only one number is given, or the second number is zero, assume that the second number has the value one.

**resume**

Continue playing the current movie after a '**pause**' or a '**stop**' command.

**segment integer [integer]**

Set the portion of the current movie that will be played to the section from the first frame number to the second. If the first number is zero and the second number is zero or less, set the portion to be the whole movie. If the second number is negative, set the portion to be from the first frame number to the end of the movie. That is, '0 0' is the whole movie, as is '0 -1', while '15 -1' is the portion from frame 15 to the end.

**start**

Move to the beginning of the current movie, make it active and start it playing.

**stop**

Stop the current movie.

**time**

Return the current frame number of the current movie.

**unload** [**movie-name**]

If no movie is specified, remove the current movie from the list of movies. Otherwise, remove the specified movie from the list of movies. **movie-name** must be a symbol, not a number.

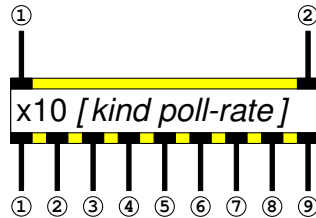
**volume** [**integer**]

Set the audio level of the current movie. The maximum level is 255; setting the level negative acts to mute the current movie, but the **'mute'** command can restore the audio level to the corresponding positive value.

**File format:** QuickTime™ movie

**Comments:** The *wqt* object was designed to address a critical weakness of the standard *movie* object: there was no way to request only a section of the movie be played, even though QuickTime™ supports this ability. One feature of the standard *movie* object was not retained—mouse motion over the *wqt* object is not detected.

## *x10*



**Description of object:** *x10* is an interface to the X-10 controller originally from X-10 Incorporated. It sends commands to a *serial* or a *serialX* object, which controls the serial port that the X-10 is attached to, and responds to data returned from the X-10 via the *serial* or *serialX* object.

**Object created:** September 1996

**Current version:** 1.0.8

**Online help file:** no

**Object theme:** Device interface

**Object class(es):** Devices

**Argument(s):**

***kind* (optional) symbol**, the type of X-10 controller to communicate with. The acceptable values are 'cm11' and 'cp290'. The default kind is 'cm11'.

***poll-rate* (optional) integer**, the rate (in milliseconds) at which the companion *serial* or *serialX* object is polled via a sample request. The default rate is 100 milliseconds between sample requests.

**Inlet(s):**

- 1 **list**, the command channel
- 2 **integer**, the feedback from the *serial* or *serialX* object

**Outlet(s):**

- 1 **integer**, the commands to the *serial* or *serialX* object
- 2 **bang**, the sample request to the *serial* or *serialX* object
- 3 **integer**, the base house-code
- 4 **bang**, the command has completed
- 5 **integer**, the device status (sent as a result of each command)
- 6 **integer**, the day number
- 7 **integer**, the hour number
- 8 **integer**, the minute number

9 **bang**, an error was detected

**Companion object(s):** works with *serialX* objects with OS 9, or *serial* objects with OS X, can be attached to *x10units* objects

**Standalone:** yes

**Retains state:** yes

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

**clear! eventNumber**

Remove the given event, **eventNumber**, where **eventNumber** is an integer between 0 and 127.

**clock!**

Set the clock of the X-10 device to match the Macintosh time-of-day clock.

**dim house-code map [level]**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, immediately dim the specified devices to the value of **level**, which is an integer between 0 and 15. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'. If **level** is not given, it is assumed to be zero.

**events?**

Interrogates the X-10 device for its event data, but does nothing with the results.

**everyday! house-code map eventNumber hourMinute function [level]**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, an event number, **eventNumber**, the encoded time, **hourMinute**, the operation to perform, **function** (either 'on', 'off' or 'dim') and the dimming level, **level**, record the event in the X-10 device as a 'normal' event for activation at the given time, every day of the week. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'. The event number is an integer between 0 and 127. The time is encoded by multiplying the desired hour by 60 and adding the minutes; it is effectively the number of minutes after midnight. The dimming level is an integer between 0 and 15.

**graphics! object-number [object-data]**

Set the graphics data of the X-10 device to the two given integer values. There is no current use for this information. If **object-data** is not given, it is assumed to be zero.

**graphics?**

Interrogates the X-10 device for its icon data, but does nothing with the results.

**housecode! letter**

Set the name of the X-10 device to the given value, **letter**, where the value is a single character symbol between 'A' and 'P'.

**housecode?**

Return the current time (day, hour and minute) and the X-10 house-code via the corresponding outlets.

**off house-code map**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, immediately turn off the specified devices. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'.

**on house-code map**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, immediately turn on the specified devices. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'.

**kind device-type**

Change the protocol to match the given **device-type**. The **device-type** is either 'cm11' or 'cp290'.

**normal! house-code map eventNumber dayMap hourMinute function [level]**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, an event number, **eventNumber**, a set of days, **dayMap**, the encoded time, **hourMinute**, the operation to perform, **function** (either 'on', 'off' or 'dim') and the dimming level, **level**, record the event in the X-10 device as a 'normal' event for activation at the given time on the given days. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'. The event number is an integer between 0 and 127. The set of days is an integer between 0 and 127, representing which days the event is to occur on, with a single bit corresponding to each day of the week. The time is encoded by multiplying the desired hour by 60 and adding the minutes; it is effectively the number of minutes after midnight. The dimming level is an integer between 0 and 15.

**reset**

Send a diagnostic command to the X-10 device, forcing a reset of the device.

**security! house-code map eventNumber dayMap hourMinute function [level]**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, an event number, **eventNumber**, a set of days, **dayMap**, the encoded time, **hourMinute**, the operation to perform, **function** (either 'on', 'off' or 'dim') and the dimming level, **level**, record the event in the X-10 device as a 'secure' event for activation at the given time on the given days. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'. The event number is an integer between 0 and 127. The set of days is an integer between 0 and 127, representing which days the event is to occur on, with a single bit corresponding to each day of the week. The time is encoded by multiplying the desired hour by 60 and adding the minutes; it is effectively the number of minutes after midnight. The dimming level is an integer between 0 and 15.

**today! house-code map eventNumber hourMinute function [level]**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, an event number, **eventNumber**, the encoded time, **hourMinute**, the operation to perform, **function** (either 'on', 'off' or 'dim') and the dimming level, **level**, record the event in the X-10 device as a 'normal' event for activation at the given time today. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'. The event number is an integer between 0 and 127. The time is encoded by multiplying the desired hour by 60 and adding the minutes; it is effectively the number of minutes after midnight. The

dimming level is an integer between 0 and 15.

**tomorrow! house-code map eventNumber hourMinute function [level]**

Given a set of unit codes for the devices controlled by the X-10 device, **map**, and the name of the X-10 device, **house-code**, an event number, **eventNumber**, the encoded time, **hourMinute**, the operation to perform, **function** (either 'on', 'off' or 'dim') and the dimming level, **level**, record the event in the X-10 device as a 'normal' event for activation at the given time tomorrow. The set of unit codes is obtained by passing the list of device numbers through the *x10units* object. The name of the X-10 device is a single character symbol between 'A' and 'P'. The event number is an integer between 0 and 127. The time is encoded by multiplying the desired hour by 60 and adding the minutes; it is effectively the number of minutes after midnight. The dimming level is an integer between 0 and 15.

**xreset**

Clear the internal state of the *x10* object, without waiting for completion of pending commands.

**Comments:**

Figure 26 shows how to connect an *x10* object to a *serial* object. Figure 27 shows how to connect an *x10* object to a *serialX* object.

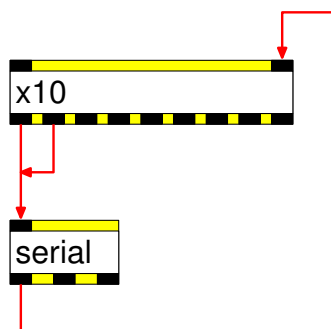


Figure 26: OS X: Connecting an *x10* object to a *serial* object

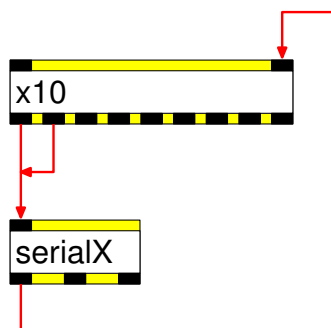
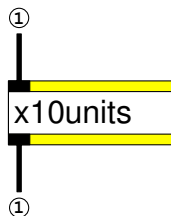


Figure 27: OS 9: Connecting an *x10* object to a *serialX* object

## *x10units*



**Description of object:** *x10units* is an auxiliary object to be used with *x10* objects. Its purpose is to permit the use of simple numbers to represent the unit codes used with the *x10* object, by mapping small integers into the bit patterns needed.

**Object created:** September 1996

**Current version:** 1.0.6

**Online help file:** no

**Object theme:** Miscellaneous

**Object class(es):** Lists

**Argument(s):** none

**Inlet(s):**

1 **integer/list/bang**, the value to be mapped

**Outlet(s):**

1 **integer**, the mapped output of the inlet value, or the previous result (if a 'bang' is received). If the inlet value is a list, the outlet value will be the bit-wise 'OR' of the result of mapping each list element.

**Companion object(s):** works with *x10* objects

**Standalone:** yes

**Retains state:** no

**Compatibility:** *Max 3.x* and *Max 4.x* {OS 9 and OS X}

**Fat, PPC-only or 68k-only:** Fat

**Command language syntax:**

*bang*

Return the previous result, if any.



## Appendix *Phidgets*

*Phidgets* are ‘plugins’ for use with the *fidget* object. They must be placed in a folder named ‘Phidgets’, located in the same folder as the *Max* program. Each plugin corresponds to a device and is named to match the device. In the descriptions of the commands for each plugin, the following abbreviations are used: ‘*standard-do*’ is the list ‘**do deviceType serialNumber/\***’, ‘*standard-get*’ is the list ‘**get deviceType serialNumber/\***’, ‘*standard-put*’ is the list ‘**put deviceType serialNumber/\***’ and ‘*standard-response*’ is the list ‘**deviceType serialNumber**’. Device serial numbers are prefixed with an underscore character (“\_”) to guarantee their interpretation as symbols rather than numbers.

The following plugins are available: *InterfaceKit004*, *InterfaceKit888*, *QuadServo*, *RFID*, *TextLCD*, *TextLCD088* and *UnitServo*.

### *InterfaceKit004*

**Description of plugin:** *InterfaceKit004* corresponds to the PhidgetInterfaceKit 0/0/4 device, which has four relay outputs.

**Plugin created:** December 2003

**Current version:** 1.0.1

#### **Command language syntax:**

##### **standard-do flip/off/on port**

Set the state of the specified digital output port to ‘1’ (‘on’), ‘0’ (‘off’) or the reverse of its current state (‘flip’). The port is limited to between 1 and 4.

##### **standard-get**

Sends the current digital output value out the report output of the *fidget* object, as the list ‘*standard-response* value bit<sub>1</sub> . . . bit<sub>4</sub>’. The value is limited to between 0 and 15.

##### **standard-put bits**

Sets the digital output to the given value. The argument **bits** is limited to between 0 and 15. Each of the digital output ports is set to the corresponding bit of the value.

### *InterfaceKit888*

**Description of plugin:** *InterfaceKit888* corresponds to the PhidgetInterfaceKit 8/8/8 device, which has eight analog inputs, eight digital inputs and eight digital outputs.

**Plugin created:** December 2003

**Current version:** 1.0.1

#### **Command language syntax:**

##### **standard-do flip/on/off port**

Set the state of the specified digital output port to ‘1’ (‘on’), ‘0’ (‘off’) or the reverse of its current state (‘flip’). The port is limited to between 1 and 8.

**standard-do trigger on/off**

Enables or disables the digital input triggering mechanism. If the mechanism is enabled, a change to one of the digital inputs results in the transmission of the current digital input through the report output of the *fidget* object. The digital input is sent as the list '*standard-response* di value bit<sub>1</sub> ... bit<sub>8</sub>'. The digital values are limited to between 0 and 255.

**standard-get ai/di/do**

Sends the current analog input ('ai'), digital input ('di') or digital output ('do') out the report output of the *fidget* object. The analog input is sent as the list '*standard-response* ai analog<sub>1</sub> ... analog<sub>8</sub>', the digital input is sent as the list '*standard-response* di value bit<sub>1</sub> ... bit<sub>8</sub>' and the digital output is sent as the list '*standard-response* do value bit<sub>1</sub> ... bit<sub>8</sub>'. The digital values are limited to between 0 and 255; the analog values are between zero and one.

**standard-put bits**

Sets the digital output to the given value. The argument **bits** is limited to between 0 and 255. Each of the digital output ports is set to the corresponding bit of the value.

## QuadServo

**Description of plugin:** *QuadServo* corresponds to the 4-Motor PhidgetServo device, which controls four servo motors.

**Plugin created:** December 2003

**Current version:** 1.0.1

**Command language syntax:****standard-get All**

Sends the current position of all the servoes (as a percentage) out the report output of the *fidget* object, as the list '*standard-response* position<sub>1</sub> position<sub>2</sub> position<sub>3</sub> position<sub>4</sub>'.

**standard-get Si servo**

Sends the current position of the specified servo (as a percentage) out the report output of the *fidget* object, as the list '*standard-response* position'. The argument **servo** is limited to between 1 and 4.

**standard-put All position<sub>1</sub> position<sub>2</sub> position<sub>3</sub> position<sub>4</sub>**

Sets the position of all the servoes to the given values, expressed as percentages of a circle.

**standard-put So servo position**

Sets the position of the specified servo to the given value, expressed as a percentage of a circle. The argument **servo** is limited to between 1 and 4.

## RFID

**Description of plugin:** *RFID* corresponds to the PhidgetRFID device, which reads RFID (Radio Frequency IDentification) tags. Note that the RFID plugin will generate messages asynchronously, when an RFID tag is brought in to close proximity of the device.

**Plugin created:** December 2003

**Current version:** 1.0.1

**Command language syntax:**

**standard-do on/off**

Asynchronous notification of RFID tags will be enabled ('on') or disabled ('off'). If asynchronous notification is enabled, when a tag is placed close to the RFID device, the list '*standard-response* RFIDtag' will be sent out the report output of the *fidget* object. The value 'RFIDtag' is a symbol composed from the hexadecimal representation of the RFID tag, prefixed with an underscore character ("\_"). Note that the list is only sent when the tag approaches the device, and a new list is output when a different tag appears.

**standard-get**

Sends the current RFID tag out the report output of the *fidget* object, as the list '*standard-response* RFIDtag'. The value 'RFIDtag' is a symbol composed from the hexadecimal representation of the RFID tag, prefixed with an underscore character ("\_").

**Comments:** Note that *RFID* can be polled (using the '**get**' command) or configured to asynchronously report the presence of RFID tags (using the '**do**' command).

## TextLCD

**Description of plugin:** *TextLCD* corresponds to the PhidgetTextLCD 20X2 device, which displays text on an LCD screen with two rows of twenty characters.

**Plugin created:** December 2003

**Current version:** 1.0.1

**Command language syntax:**

**standard-do backlight on/off**

Turns the LCD screen backlight on or off.

**standard-do clear [1/2]**

Clears the top row of the LCD screen ('1'), the bottom row ('2') or both rows, if no argument is given.

**standard-do cursor left/right**

Move the cursor of the LCD screen left or right.

**standard-do display left/right**

Shift the text shown on the LCD screen either left or right.

**standard-do entrymode [reverse] [shift]**

Specify the text entry mode of the LCD screen. If 'reverse' is specified, the text entry position decreases after each character is added to the screen. If 'reverse' is not specified, the text entry position increases after each character is added to the screen. If 'shift' is specified, the screen is shifted after each character is added to the screen, in the direction opposite to the text entry. The entered text appears to stay still while the rest of the screen moves. If 'shift' is not specified, the screen is not shifted.

**standard-do go row [column]**

Move the text entry point for the LCD screen to the given row (either '1' or '2') and the given column (between 1 and 20). If the argument **column** is not present, it is assumed to be one.

**standard-do off**

Turn off the LCD screen.

**standard-do on [blink] [cursor]**

Turn on the LCD screen. The cursor format is indicated by the keywords 'blink' and 'cursor'. If 'blink' is specified, a block cursor will be used (the whole text cell at the cursor location will blink off and on). If 'cursor' is specified, an underline cursor will be used. Both 'blink' and 'cursor' can be specified.

**standard-do write anything**

Display the given values as text on the LCD screen.

*TextLCD088*

**Description of plugin:** *TextLCD088* corresponds to the PhidgetTextLCD 20X2 with 0/8/8 InterfaceKit device, which displays text on an LCD screen with two rows of twenty characters, as well as providing eight digital inputs and eight digital outputs.

**Plugin created:** December 2003

**Current version:** 1.0.1

**Command language syntax:**

**standard-do flip/on/off port**

Set the state of the specified digital output port to '1' ('on'), '0' ('off') or the reverse of its current state ('flip'). The port is limited to between 1 and 8.

**standard-do to backlight on/off**

Turns the LCD screen backlight on or off.

**standard-do to clear [1/2]**

Clears the top row of the LCD screen ('1'), the bottom row ('2') or both rows, if no argument is given.

**standard-do to cursor left/right**

Move the cursor of the LCD screen left or right.

**standard-do to display left/right**

Shift the text displayed on the LCD screen either left or right.

**standard-do to entrymode [reverse] [shift]**

Specify the text entry mode of the LCD screen. If 'reverse' is specified, the text entry position decreases after each character is added to the screen. If 'reverse' is not specified, the text entry position increases after each character is added to the screen. If 'shift' is specified, the screen is shifted after each character is added to the screen, in the direction opposite to the text entry. The entered text appears to stay still while the rest of the screen moves. If 'shift' is not specified, the screen is not shifted.

**standard-do to go row [column]**

Move the text entry point for the LCD screen to the given row (either '1' or '2') and the given column (between 1 and 20). If the argument **column** is not present, it is assumed to be one.

**standard-do to off**

Turn off the LCD screen.

**standard-do to on [blink] [cursor]**

Turn on the LCD screen. The cursor format is indicated by the keywords 'blink' and 'cursor'. If 'blink' is

specified, a block cursor will be used (the whole text cell at the cursor location will blink off and on). If 'cursor' is specified, an underline cursor will be used. Both 'blink' and 'cursor' can be specified.

**standard-do trigger on/off**

Enables or disables the digital input triggering mechanism. If the mechanism is enabled, a change to one of the digital inputs results in the transmission of the current digital input through the report output of the *fidget* object. The digital input is sent as the list '*standard-response* di value bit<sub>1</sub> ... bit<sub>8</sub>'. The digital values are limited to between 0 and 255.

**standard-do to write anything**

Display the given values as text on the LCD screen.

**standard-get di/do**

Sends the current digital input ('di') or digital output ('do') out the report output of the *fidget* object. The digital input is sent as the list '*standard-response* di value bit<sub>1</sub> ... bit<sub>8</sub>' and the digital output is sent as the list '*standard-response* do value bit<sub>1</sub> ... bit<sub>8</sub>'. The digital values are limited to between 0 and 255.

**standard-put bits**

Sets the digital output to the given value. The argument **bits** is limited to between 0 and 255. Each of the digital output ports is set to the corresponding bit of the value.

## UnitServo

**Description of plugin:** *UnitServo* corresponds to the 1-Motor PhidgetServo device, which controls a single servo motor.

**Plugin created:** December 2003

**Current version:** 1.0.1

**Command language syntax:****standard-get**

Sends the current position of the servo (as a percentage) out the report output of the *fidget* object, as the list '*standard-response* position'.

**standard-put position**

Sets the position of the servo to the given value, expressed as a percentage of a circle.

## Index

### APL

- decode, 97
- drop, 99
- encode, 100
- reduce, 111
- reshape, 104
- reverse, 112
- rotate, 113
- scan, 116
- take, 121

### Objects

- bqt, ii, viii, ix, 1
- caseShift, ii, vi, viii, 4
- changes, ii, vi, viii, 5
- coll, 95
- compares, ii, viii, 6
- dataType, ii, viii, 7
- drop, vi
- fidget, ii, v, viii, 8, 134–136, 138
- fileLogger, ii, v, viii, 10
- gcd, ii, viii, 11
- gvp, ii, v, viii, ix, 12, 67
- jet, vi
- ldp1550, ii, v, viii, 17
- length, vi
- listen, ii, vi, viii, 21
- listType, ii, v, vi, viii, 26
- map1d, ii, v, vi, viii, 27
- map2d, ii, v, vi, viii, 32
- map3d, ii, v, vi, viii, 37
- memory, ii, viii, ix, 42
- movie, 1, 3, 126, 128
- mtc, ii, v, vi, viii, 44, 48, 49, 67, 104
- mtcTrack, ii, vi, viii, 45, 48
- not, 50
- notX, ii, vi, viii, 50
- pfsM, ii, v, viii, ix, 51
- queue, ii, vi, viii, 57
- rcx, ii, v, viii, 59
- segment, vi
- selectX, viii
- sense, 64
- senseX, ii, v, 63

- serial, iv, v, 12, 13, 15–20, 44–46, 59, 65, 67, 69, 70, 129, 130, 132
- serialX, ii, iv–vi, viii, 12, 13, 15–20, 44–46, 59, 65, 69–71, 129, 130, 132
- shotgun, ii, v, viii, 68
- spaceball, ii, v, vi, viii, 69
- speak, ii, v, vi, viii, 72
- stack, ii, viii, ix, 75
- sysLogger, ii, v, viii, 77
- table, 43, 53
- take, vi
- tcpClient, ii, v, vi, viii, 78, 82–88, 90
- tcpLocate, ii, vi, viii, 82
- tcpMultiServer, ii, v, vi, viii, 78, 79, 81–83, 85, 88, 90
- tcpServer, ii, v, vi, viii, 78, 79, 81, 82, 86, 90
- udpPort, ii, v, viii, 85, 88–90
- Vabs, ii, vi, viii, 91
- Vassemble, ii, v, 92
- Vceiling, ii, vi, viii, 93
- Vcollect, ii, v, viii, 94
- Vcos, ii, vi, viii, 96
- Vdecode, ii, v, 97
- Vdistance, iii, vi, viii, 98
- Vdrop, iii, vi, viii, 99, 119
- Vencode, iii, v, 100
- Vexp, iii, vi, viii, 101
- Vfloor, iii, vi, viii, 102
- Vinvert, iii, vi, viii, 103
- Vjet, iii, vi, viii, 104
- Vlength, iii, vi, viii, 105
- Vlog, iii, vi, viii, 106
- Vltrim, iii, v, 107
- Vmean, iii, vi, viii, 108
- Vnegate, iii, vi, viii, 110
- Vreduce, iii, vi, viii, 111
- Vreverse, iii, vi, viii, 112
- Vrotate, iii, vi, viii, 113
- Vround, iii, vi, viii, 114
- Vrtrim, iii, v, 115
- Vscan, iii, vi, viii, 116
- Vsegment, iii, vi, viii, 117
- Vsin, iii, vi, viii, 118
- Vsplit, iii, v, viii, 119

Vsqr, [iii](#), [vi](#), [viii](#), [120](#)  
 Vtake, [iii](#), [vi](#), [viii](#), [119](#), [121](#)  
 Vtokenize, [iii](#), [v](#), [122](#)  
 Vtrim, [iii](#), [v](#), [123](#)  
 Vtruncate, [iii](#), [vi](#), [viii](#), [124](#)  
 Vunspell, [iii](#), [v](#), [125](#)  
 wqt, [iii](#), [viii](#), [126](#)  
 x10, [iii](#), [v](#), [viii](#), [ix](#), [129](#), [133](#)  
 x10units, [iii](#), [viii](#), [130–133](#)

## Phidgets

InterfaceKit004, [134](#)  
 InterfaceKit888, [134](#)  
 QuadServo, [134](#), [135](#)  
 RFID, [134–136](#)  
 TextLCD, [134](#), [136](#)  
 TextLCD088, [134](#), [137](#)  
 UnitServo, [134](#), [138](#)

## Themes

Device interface  
   fidget, [8](#)  
   gvp, [12](#)  
   ldp1550, [17](#)  
   listen, [21](#)  
   mtc, [44](#)  
   rcx, [59](#)  
   serialX, [65](#)  
   spaceball, [69](#)  
   speak, [72](#)  
   x10, [129](#)

## Miscellaneous

caseShift, [4](#)  
 dataType, [7](#)  
 fileLogger, [10](#)  
 gcd, [11](#)  
 listType, [26](#)  
 mtcTrack, [48](#)  
 notX, [50](#)  
 shotgun, [68](#)  
 sysLogger, [77](#)  
 Vabs, [91](#)  
 Vceiling, [93](#)  
 Vcollect, [94](#)  
 Vcos, [96](#)  
 Vdistance, [98](#)

Vexp, [101](#)  
 Vfloor, [102](#)  
 Vinvert, [103](#)  
 Vlog, [106](#)  
 Vnegate, [110](#)  
 Vround, [114](#)  
 Vsin, [118](#)  
 Vsqr, [120](#)  
 Vtruncate, [124](#)  
 x10units, [133](#)

## Programming aids

changes, [5](#)  
 compares, [6](#)  
 dataType, [7](#)  
 map1d, [27](#)  
 map2d, [32](#)  
 map3d, [37](#)  
 memory, [42](#)  
 pfs, [51](#)  
 queue, [57](#)  
 senseX, [63](#)  
 stack, [75](#)

## QuickTime™

bqt, [1](#)  
 wqt, [126](#)

## TCP/IP

tcpClient, [78](#)  
 tcpLocate, [82](#)  
 tcpMultiServer, [83](#)  
 tcpServer, [86](#)  
 udpPort, [89](#)

## Vector manipulation

Vabs, [91](#)  
 Vassemble, [92](#)  
 Vceiling, [93](#)  
 Vcollect, [94](#)  
 Vcos, [96](#)  
 Vdecode, [97](#)  
 Vdistance, [98](#)  
 Vdrop, [99](#)  
 Vencode, [100](#)  
 Vexp, [101](#)  
 Vfloor, [102](#)  
 Vinvert, [103](#)  
 Vjet, [104](#)  
 Vlength, [105](#)

Vlog, 106  
 Vltrim, 107  
 Vmean, 108  
 Vnegate, 110  
 Vreduce, 111  
 Vreverse, 112  
 Vrotate, 113  
 Vround, 114  
 Vrtrim, 115  
 Vscan, 116  
 Vsegment, 117  
 Vsin, 118  
 Vsplit, 119  
 Vsqrt, 120  
 Vtake, 121  
 Vtokenize, 122  
 Vtrim, 123  
 Vtruncate, 124  
 Vunspell, 125  
 Vround, 114  
 Vscan, 116  
 Vsin, 118  
 Vsqrt, 120  
 Vtruncate, 124

## Vectors

### Dyadic operations

caseShift, 4  
 Vdecode, 97  
 Vdrop, 99  
 Vencode, 100  
 Vjet, 104  
 Vrotate, 113  
 Vsegment, 117  
 Vsplit, 119  
 Vtake, 121

### Monadic operations

notX, 50  
 Vabs, 91  
 Vceiling, 93  
 Vcos, 96  
 Vdistance, 98  
 Vexp, 101  
 Vfloor, 102  
 Vinvert, 103  
 Vlength, 105  
 Vlog, 106  
 Vmean, 108  
 Vnegate, 110  
 Vreduce, 111  
 Vreverse, 112